Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

ESTIMACIÓN DEL NIVEL DE IMPUREZAS EN MUESTRAS DE ACEITE

Alumno/a: **Muñoz Castillo, Ana**

Tutor/a: Prof. D. Pablo Cano Marchal
Prof. Dª. Silvia Satorres Martínez
Dpto.: Departamento de Ingeniería Electrónica y Automática

Septiembre, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Electrónica y Automática

Don PABLO CANO MARCHAL, tutor del Trabajo Final de Máster titulado: ESTIMACIÓN DEL NIVEL DE IMPUREZAS EN MUESTRAS DE ACEITE, que presenta ANA MUÑOZ CASTILLO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2021

El alumno:	El tutor:	La tutora:
MUÑOZ CASTILLO ANA - 77380398A Firmado digitalmente por MUÑOZ CASTILLO ANA - 77380398A Fecha: 2021.09.06 16:51:43 +02'00'	CANO MARCHAL PABLO - 77344429Y Firmado digitalmente por CANO MARCHAL PABLO - 77344429Y Fecha: 2021.09.06 17:26:17 +02'00'	SATORRES MARTINEZ SILVIA - 26741651B Firmado digitalmente por SATORRES MARTINEZ SILVIA - 26741651B Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26741651B, givenName=SILVIA, sn=SATORRES MARTINEZ, cn=SATORRES MARTINEZ SILVIA - 26741651B Fecha: 2021.09.06 17:09:10 +02'00'
ANA MUÑOZ CASTILLO	PABLO CANO MARCHAL	SILVIA SATORRES MARTÍNEZ

RESUMEN

La producción de aceite de oliva virgen (AOV) es una importante actividad económica que debe mantener su competitividad en un entorno de economía global. Para ello son necesarias nuevas técnicas que permitan el control a tiempo real de los parámetros de calidad en la extracción del aceite. Uno de los aspectos que determinan esta calidad, es la presencia de impurezas insolubles. En este contexto, el objetivo principal de este trabajo fue el desarrollo e implementación de un sistema de visión por computador para determinar el nivel de impurezas presentes en muestras de aceite de oliva virgen en condiciones de laboratorio. Para ello se estudiaron cuatro vectores de entrada diferentes, derivados del histograma de los canales de los espacios de color RGB, HSV y CIELAB. Además, se aplicó un método de extracción de características antes de la clasificación. Los clasificadores utilizados fueron las máquinas de soporte vectorial (Support Vector Machines o SVMs), la regresión logística multinomial y el clasificador Random Forest. El mejor resultado de clasificación se logró utilizando un método de extracción de características *Kernel Principal Component Analysis* (KPCA) junto a un clasificador Random Forest, con una precisión del 65,45%. Aunque esta cifra es inferior a la de otros trabajos de similar naturaleza, en general los resultados avalan la viabilidad de emplear un sistema basado en visión artificial y reconocimiento de patrones para realizar una clasificación aproximada y rápida del contenido de las muestras de aceite de oliva.

ABSTRACT

The production of Virgin Olive Oil (VOO) is an important economic activity that must remain competitive in a global economic environment. This requires new techniques that allow real-time control of quality parameters in oil extraction. One of the aspects that determine this quality is the content of insoluble impurities. In this context, the main objective of this work was the development and implementation of a computer vision system classify the content of impurities in virgin olive oil samples under laboratory conditions. For this purpose, three different input vectors constructed from the histograms of the channels of the RGB, HSV and CIELAB color spaces were studied. Besides, a feature extraction was performed before classification. The classifiers used were Support Vector Machines (SVMs), multinomial logistic regression and the Random Forest classifier. The best classification result was achieved using a Kernel Principal Component Analysis (KPCA) jointly with a Random Forest classifier, with an accuracy of 65.45%. Although this accuracy is lower than the one obtained in other similar works, overall, the results support the feasibility of using a system based on computer vision and pattern recognition to perform an approximate and fast classification of the content of olive oil samples.

ÍNDICE DE CONTENIDO

1 Introducción 1
1.1 Justificación y planteamiento del problema 1
1.2 Estado del arte 5
1.3 Objetivos 11
1.4 Proceso de desarrollo 11
1.5 Estructura de la memoria 12
2 Fundamentos teóricos y herramientas de software 14
2.1 Inteligencia artificial y aprendizaje automático 14
2.2 Método de clasificación 15
2.3 Herramientas de software 18
2.3.1 Lenguaje de programación. Python 18
2.3.2 OpenCV 19
2.3.3 Scikit-Learn 20
2.3.4 NumPy 22
2.3.5 Matplotlib 23
3 Obtención de imágenes 24
4 Obtención del vector de características 26
4.1 Segmentación de imágenes 27
4.2 Composición del vector de características 30
4.3 Datos extraídos 32
4.4 Normalización 36
4.5 Extracción de características 36
4.5.1 Visualización de la separación entre clases 37
5 Obtención del clasificador 39
5.1 Diseño del clasificador 39
5.1.1 Máquinas de soporte vectorial 41
5.1.2 Random Forest 45
5.1.3 Regresión logística 47
5.2 Clasificador óptimo 48
6 Resultados 51
6.1 Imágenes de muestra 51
6.2 Vector de características 52
6.2.1 Resultados de la segmentación de imágenes 52
6.2.2 Composición del vector de características. Visualización de histogramas. 54
6.2.3 Vectores de características configurados 59
6.2.4 Resultado de la normalización del vector de características 63
6.2.5 Extracción de características. Visualización de la separación entre clases 64
6.3 Clasificador 68
6.3.1 Coeficientes de la transformación PCA 68
6.3.2 Hiperparámetros obtenidos 77
6.3.2.1 Máquina de Soporte Vectorial Lineal 77
6.3.2.2 Máquina de Soporte Vectorial: kernel RBF 82
6.3.2.3 Máquina de Soporte Vectorial: kernel polinómico 88
6.3.2.4 Clasificador Random Forest 95
6.3.2.5 Regresión logística 100
6.3.3 Clasificador óptimo 105
7 Interfaz gráfica de usuario 115

8 Discusión y conclusión 118
9 Referencias bibliográficas 121
10 Anexos 127
10.1 Anexo I: Ficheros relacionados con la obtención del vector de características 127
10.1.1 vector.py 127
10.1.2 HistFunction.py 131
10.1.3 PairPlot_PCA.py 143
10.1.4 DataBase_HSab_Inicial.py 146
10.1.5 DataBase_HSab.py 148
10.1.6 DataBase_GRLab.py 150
10.1.7 DataBase_GRL.py 153
10.2 Anexo II: Ficheros relacionados con el ajuste de hiperparámetros 156
10.2.1 LinearSVC_nestedCV.py 156
10.2.2 SVC_rbf_nestedCV 158
10.2.3 SVC_poly_nestedCV 163
10.2.4 RandomForest_PatameterGrid.py 167
10.2.5 RandomForest_HSab.py 169
10.2.6 RandomForest_GRL.py 171
10.2.7 LogisticRegression.py 173
10.3 Anexo III: Ficheros relacionados con el clasificador óptimo y la interfaz gráfica de usuario 176
10.3.1 OptimalClassifier_HSab_inicial.py 176
10.3.2 OptimalClassifier_HSab.py 177
10.3.3 OotimalClassifier_GRLab.py 179
10.3.4 OptimalClassifier_GRL.py 181
10.3.5 Classifier_RandomForest.py 184
10.3.6 Classifier_SVM.py 186
10.3.7 Classifier.py 188
10.3.8 GUI.py 189

ÍNDICE DE FIGURAS

Figura 1. Esquema clásico de una almazara para la elaboración de aceite de oliva virgen (Cano Marchal et al., 2011). 3

Figura 2. Configuración de adquisición de imágenes, compuesta por el anillo de iluminación y la cámara (parte superior). En la parte inferior de la imagen está el crisol de porcelana que contiene la muestra de aceite de oliva (Cano Marchal, et al. 2013). 24

Figura 3. Máscara de segmentación de la muestra 10_10__0 28

Figura 4. Imagen de muestra i0_03__28 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.03% 30

Figura 5. Construcción del vector de entrada final a partir del histograma original de un canal. El primer subconjunto del vector son los valores del histograma en orden descendente (Cano Marchal, et al. 2013). 31

Figura 6. Esquema del proceso de obtención de la matriz de características 33

Figura 7. Esquema del proceso de obtención de la matriz de características siguiendo la configuración propuesta por Cano Marchal, et al. (2013) 35

Figura 8. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características HSab Inicial. 38

Figura 10. Esquema del diseño de clasificadores mediante validación cruzada anidada. 41

Figura 10. Esquema de la metodología para la obtención del clasificador óptimo. Las etapas en azul se corresponden al proceso descrito en el apartado 3.1, mientras que las etapas en verde se corresponden con el procedimiento expuesto en este apartado. 49

Figura 11. Diferentes muestras de aceite de oliva tras el proceso de secado junto con su contenido de impurezas donde i denota el porcentaje de impurezas en las muestras. 51

Figura 12. Distribución del contenido de impurezas no solubles para las muestras estudiadas. 52

Figura 13. Imagen de muestra i0_04__21 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.04% 53

Figura 14. Imagen de muestra i0_04__21 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.04% 53

Figura 15. Imagen de muestra i0_05__2 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.05% 53

Figura 16. Imagen de muestra i0_06__12 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.06% 54

Figura 17. Imagen de muestra i0_10__0 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.10% 54

Figura 18. Histogramas agrupados de los canales H, S, a y b de las imágenes de muestra segmentadas. 55

Figura 19. Histogramas agrupados de los canales del espacio de color RGB de las imágenes de muestra segmentadas. 56

Figura 20. Histogramas agrupados de los canales del espacio de color HSV de las imágenes de muestra segmentadas. 56

Figura 21. Histogramas sin agrupar de los canales del espacio de color CIELAB de las imágenes de muestra segmentadas. 57

Figura 22. Ampliación de los histogramas agrupados de los canales del espacio de color RGB de las imágenes de muestra segmentadas..... 57

Figura 23. Ampliación de los histogramas agrupados de los canales del espacio de color HSV de las imágenes de muestra segmentadas. 58

Figura 24. Ampliación de los histogramas agrupados de los canales del espacio de color CIELAB de las imágenes de muestra segmentadas..... 58

Figura 25. Vector de características HSab Inicial, formado por los canales H, S, a y b transformados según el algoritmo definido en la figura 5 59

Figura 27. Vector de características HSab, formado por los canales H, S, a y b..... 60

Figura 28. Vector de características GRLab, formado por los canales G, R, L, a y b.. 61

Figura 29. Vector de características GRL, formado por los canales G, R y L..... 62

Figura 29. Vector de características GRL normalizado. 63

Figura 30. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características HSab..... 65

Figura 31. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características GRLab. 66

Figura 32. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características GRL. 67

Figura 33. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características HSab Inicial..... 69

Figura 34. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características HSab Inicial..... 70

Figura 35. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características HSab. 71

Figura 36. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características HSab. 72

Figura 37. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características GRLab. 73

Figura 38. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características GRLab. 74

Figura 39. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características GRL. 75

Figura 40. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características GRL. 76

Figura 41. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características HSab Inicial. 78

Figura 42. . Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características HSab. 79

Figura 43. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características GRLab. 80

Figura 44. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características GRL. . 81

Figura 45. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características HSab Inicial. 83

Figura 46. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características HSab. ... 84

Figura 47. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características GRLab.. 85

Figura 48. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características GRL. 86

Figura 49. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características HSab Inicial. 90

Figura 50. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características HSab. 91

Figura 51. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características GRLab. 92

Figura 52. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características GRL. 93

Figura 53. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características HSab Inicial. 101

Figura 54. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características HSab..... 102

Figura 55. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características GRLab. 103

Figura 56. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características GRL. 104

Figura 57. Coeficientes asociados con cada componente PCA para la Máquina de Soporte Vectorial Lineal con vector de características GRLab. 108

Figura 58. Coeficientes asociados con cada componente PCA para la Máquina de Soporte Vectorial Lineal con vector de características GRL. 109

Figura 59. Coeficientes asociados con cada componente PCA para la Regresión Logística Multinomial con vector de características GRLab. 110

Figura 60. Coeficientes asociados con cada componente PCA para la Regresión Logística Multinomial con vector de características GRL. 111

Figura 61. Coeficientes asociados con cada componente PCA para el clasificador Random Forest con vector de características HSab. 112

Figura 62. Coeficientes asociados con cada componente PCA para el clasificador Random Forest con vector de características GRL. 112

ÍNDICE DE TABLAS

Tabla 1. Muestras de aceite perteneciente a cada clase (i denota el porcentaje de impurezas de la muestra) 25

Tabla 2. Valores máximos y mínimos para el rango de segmentación 28

Tabla 3. Valores asignados al hiperparámetro C para el clasificador SVM lineal. 43

Tabla 4. Valores asignados los hiperparámetros C y gamma para el clasificador SVM con kernel RBF. 44

Tabla 5. Valores asignados los hiperparámetros C y degree para el clasificador SVM con kernel polinómico. 45

Tabla 6. Valores asignados a los hiperparámetros del clasificador Random Forest. ... 46

Tabla 7. Valores asignados al hiperparámetro C para la regresión logística multinomial. 48

Tabla 8. Parámetros de agrupación establecidos para cada canal en las distintas pruebas realizadas. 55

Tabla 9. Vectores de características evaluados. 63

Tabla 10. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características HSab Inicial. 78

Tabla 11. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características HSab 79

Tabla 12. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características GRLab. 80

Tabla 13. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características GRL. 81

Tabla 14. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab Inicial. 82

Tabla 15. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab. 84

Tabla 16. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRLab. 85

Tabla 17. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRL. 86

Tabla 18. Valores obtenidos de los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab Inicial. 87

Tabla 19. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab. 87

Tabla 20. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRLab. 88

Tabla 21. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRL. 88

Tabla 22. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab Inicial. 89

Tabla 23. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab. 91

Tabla 24. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRLab. 92

Tabla 25. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRL. 93

Tabla 26. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab Inicial. 94

Tabla 27. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab. 94

Tabla 28. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRLab. 95

Tabla 29. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRL. 95

Tabla 30. Valores por defecto de los hiperparámetros del clasificador Random Forest. 96

Tabla 31. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características HSab Inicial. 96

Tabla 32. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características HSab Inicial. 97

Tabla 33. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características HSab. 97

Tabla 34. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características HSab. 97

Tabla 35. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características GRLab. 98

Tabla 36. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características GRLab. 98

Tabla 37. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características GRL. 98

Tabla 38. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características GRL. 99

Tabla 39. Valores asignados a los hiperparámetros del clasificador Random Forest en la búsqueda de rejilla 99

Tabla 40. Modelos óptimos obtenidos para el clasificador Random Forest con los vectores de características HSab y GRL. 99

Tabla 41. Precisiones de entrenamiento y validación obtenidas con el modelo óptimo del clasificador Random Forest para los vectores de características HSab y GRL. .. 100

Tabla 42. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características HSab Inicial. 101

Tabla 43. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características HSab. 102

Tabla 44. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características GRLab. 103

Tabla 45. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características GRL. 104

Tabla 46. Resumen de características de los clasificadores considerados como clasificador óptimo. 105

Tabla 47. Resumen de los resultados obtenidos. 106

Tabla 48. Modelo óptimo obtenido para el clasificador Random Forest con el vector de características HSab y un método de extracción de características KPCA. 113

Tabla 49. Resumen de resultados en el estudio de mejora del clasificador óptimo. .. 114

ÍNDICE DE CUADROS

Cuadro 1. Función auxiliar para la agrupación de histogramas 26

Cuadro 2. Función que obtiene el histograma agrupado de una imagen 27

Cuadro 3. Función que segmenta una imagen dada y aplica una transformación morfológica de apertura y cierre 29

Cuadro 4. Función que representa los histogramas agrupados de las imágenes de muestra para tres espacios de color 31

Cuadro 5. Función que representa los histogramas agrupados de los canales de las imágenes segmentadas para tres espacios de color 32

Cuadro 6. Función que calcula el vector de características siguiendo la configuración propuesta por Cano Marchal, et al. (2013) 35

Cuadro 7. Función que carga el modelo entrenado. 116

Cuadro 8. Función que elabora el vector de características de una imagen proporcionada. 116

Cuadro 9. Función que realiza el preprocesamiento de los datos de entrada al clasificador 116

Cuadro 10. Función que devuelve la predicción del nivel de impurezas de una muestra de aceite de oliva 117

1 Introducción

1.1 Justificación y planteamiento del problema

En los últimos años ha aumentado la exigencia de los estándares de calidad y seguridad en la industria agroalimentaria y el sector olivarero no ha sido ajeno a este proceso (Parra-López et al., 2016). La calidad del aceite de oliva virgen (AOV) se evalúa de acuerdo con normas internacionales que establecen los límites de sus propiedades químicas y organolépticas y, conforme a ello, se clasifica y etiqueta, siendo el aceite de olive virgen extra (AOVE) la distinción de mayor calidad. En paralelo, los consumidores demandan mejores productos en términos de seguridad, salud y propiedades organolépticas, además de precios competitivos. Tanto es así que de su satisfacción depende, no solo la supervivencia, sino también la competitividad de las empresas del sector (Parra-López et al., 2016).

La producción de AOV es una importante actividad económica, originaria de los países mediterráneos, que se ha duplicado en las dos últimas décadas, en parte por la extensión de esta producción al resto del mundo (Beltrán-Ortega, et al., 2016). Para el año 2019, la FAOSTAT (base de datos estadísticos de la Organización de las Naciones Unidas para la Alimentación y la Agricultura) sitúa a España como el primer país a nivel internacional en extensión y producción de aceituna y el mayor exportador de AOV, seguido de lejos por Italia, Portugal y Túnez. Por otro lado, Parra-López et al. (2016) afirman que Andalucía era en 2013 la región española que concentraba la mayor extensión de este cultivo, representando el 75,7% de la producción de aceituna y el 62,3% de la superficie de este cultivo en España. Dada la importancia de este sector, parece necesario mantener la competitividad en un escenario internacional, no solo en términos de cantidad de producción y exportación, sino también en lo que se refiere a la calidad del producto final.

Tal y como describen Cano Marchal et al. (2011), una fábrica de aceite de oliva virgen (almazara) suele tener tres zonas diferenciadas: un patio de recepción de la aceituna, las instalaciones de extracción del aceite y la bodega donde se almacena. El proceso de elaboración comienza en el patio de recepción, donde los agricultores depositan las aceitunas en la tolva. Posteriormente, el producto se desplaza por medio de cintas transportadoras a la máquina de lavado industrial, donde se eliminan las piedras y el barro. Una vez limpia, la aceituna se muele para formar una pasta que posteriormente pasa a una batidora en la que se calienta para reducir su viscosidad. Mientras, unas palas la mueven lentamente para permitir la agregación de las gotas de aceite y la ruptura de los microgeles, formados por restos de tejidos y agua, que retienen pequeñas gotas de aceite. Una vez batida, la pasta se inyecta en el decánter, una centrífuga horizontal en la que se separa el aceite de orujo debido a la diferencia de densidades entre ambos componentes. El aceite que sale del decánter presenta un nivel elevado de humedad e impurezas sólidas que afectan negativamente a su conservación, por lo que es necesario separar estos elementos del aceite antes de su almacenamiento en bodega. Para este proceso, el aceite puede centrifugarse en una centrífuga vertical con adición de agua o decantarse por acción de la gravedad en depósitos de acero inoxidable diseñados para este fin. La figura 1 muestra un esquema del proceso de elaboración del AOV según lo describió Cano Marchal et al. (2011).

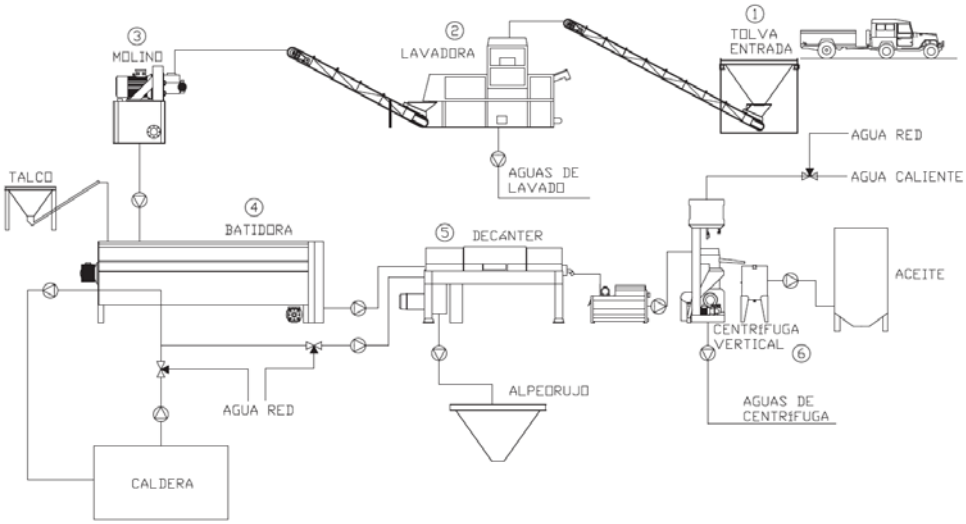


Figura 1. Esquema clásico de una almazara para la elaboración de aceite de oliva virgen (Cano Marchal et al., 2011).

Como queda patente, el AOV se extrae exclusivamente por procesos mecánicos y físicos que no conducen a alteraciones en su composición (IOC, 2021) . Es por esto por lo que su calidad está directamente relacionada con las propiedades de la aceituna, las variables del proceso de extracción y las condiciones de almacenaje. Aunque muchas de las variables relacionadas con el primer factor no son controlables, pues dependen de las condiciones de cultivo de la aceituna, las variables del proceso de producción y el almacenaje sí pueden controlarse de forma precisa (Kalogianni et al., 2019). No obstante, es muy frecuente todavía en la actualidad que sea el maestro molinero el que ajuste manualmente las variables básicas que intervienen en el proceso de extracción del aceite, en ocasiones según sus propios conocimientos, y la evaluación de la calidad del aceite la haga un laboratorio externo, con la consiguiente demora en la obtención de resultados (Cano-Marchal et al., 2011). Un control adecuado de estas variables podría dar lugar a una optimización del rendimiento, la economía y el impacto ambiental de la producción de calidad del aceite. Sin embargo, si los métodos y las variables de procesamiento no se seleccionan adecuadamente, la calidad del aceite de oliva puede verse afectada, incluso si

la materia prima es de buena calidad (Kalogianni, et al., 2019). Sin duda, es un hecho que hace necesario encontrar procedimientos integrados que permitan al operador de fábrica tomar decisiones a tiempo real.

El reglamento CE 61/2011 y el Consejo Oleícola Internacional caracterizan calidad del aceite comercial según varios parámetros relacionados con el estado de deterioro del AOV. Estos parámetros incluyen la acidez libre y otros relacionados con su estado de oxidación, como el índice de peróxidos y los índices de absorción K_{232}, K_{270} y ΔK (Martínez Gila et al., 2018). Otros compuestos menores, como los fenoles, también están relacionados con la calidad y los atributos sensoriales del aceite de oliva y juegan un papel importante en las propiedades positivas para la salud del aceite de oliva virgen (López-Miranda et al., 2010).

Además, otro de los aspectos que determinan la calidad del AOV es la presencia de impurezas insolubles (IOC, 2021) que suelen ser restos de sólidos orgánicos de la propia aceituna. La determinación del nivel de estas impurezas es un análisis muy común, al igual que la determinación del contenido de humedad (Gila et al., 2020). Estos análisis se tienen en cuenta en la comercialización del aceite de oliva, incluso en las transacciones a granel, ya que, si el producto contiene niveles de humedad e impurezas insolubles superiores a los establecidos, el precio se reduce (Cano-Marchal et al., 2013). Pero, además, conocer rápida y eficazmente el valor de estos parámetros también es útil durante el proceso de elaboración del aceite de oliva, ya que permite al operador verificar el contenido de humedad e impurezas insolubles en diferentes puntos del proceso. De esta forma, puede conocerse su contenido en el mosto aceitoso del decantador horizontal, comprobar el rendimiento de eliminación de los sistemas de clarificación, como la centrifugadora vertical o los tanques de sedimentación, o conocer la cantidad de aceite desperdiciado durante las purgas realizadas en los tanques de sedimentación y almacenamiento (Gila et al., 2020).

El método de determinación del contenido de impurezas insolubles establecido en la norma internacional ISO 663:2017 requiere tratar la porción para análisis con un exceso de n-hexano o de éter de petróleo para después filtrar la disolución resultante. A continuación, se lavan tanto el filtro como el residuo con el mismo disolvente, se deja evaporar al aire la mayor parte del disolvente y se completa la evaporación en la estufa conectada a 103 °C y después se pesa (ISO, 2017). No obstante, este procedimiento es bastante laborioso, y sería conveniente tener un método alternativo aproximado para evaluar este proceso de eliminación de impurezas.

En este contexto, y dada la necesidad de mantener la competitividad del sector del aceite de oliva en un entorno cada vez más exigente con el producto final, son necesarias nuevas técnicas que permitan el control a tiempo real de los parámetros de calidad en la extracción del aceite. En este sentido, la visión por computador proporciona una automatización de este control, de forma no invasiva y rentable, cumpliendo con los parámetros establecidos en la normativa y los estándares actuales de producción de un aceite de oliva de calidad (Guzmán et al., 2013b).

1.2 Estado del arte

En los últimos años, la aplicación de la visión por computador en la industria agroalimentaria ha aumentado considerablemente. Se han desarrollado técnicas innovadoras para el control, la inspección y la clasificación de procesos con el fin de lograr los requisitos de calidad y seguridad exigidos por el mercado. Sin embargo, no existen muchas aplicaciones de la visión por computador para la industria del aceite de oliva.

Algunos métodos desarrollados para el control de calidad del aceite de oliva tienen elevados costes de adquisición y mantenimiento (Buratti et al., 2018). Por el contrario, las tecnologías basadas en imagen digital son una alternativa prometedora para el análisis de alimentos y otros productos al ser rápidas,

económicas y no invasivas. Algunos estudios demuestran su viabilidad en alimentos como el plátano (Dittakan et al., 2017; Helwan et al., 2021; Olaniyi et al., 2017; Sanaeifar et al., 2016), las naranjas (Javadikia et al., 2018; Sabzi et al., 2017), el arroz blanco (Zareiforoush et al., 2015) y el jamón curado (Muñoz et al., 2015), entre otros. Muchos otros estudios se centran en la clasificación de frutas (Garcia et al., 2016; Lu et al., 2016; Nasir et al., 2021; Wang et al., 2015; Zhang et al., 2014; Zhang et al., 2016). La mayoría de estos estudios utilizan información de modelos de color asociados con técnicas de reconocimiento de patrones para agrupar, o clasificar, las muestras en categorías según estándares de similitud.

Así mismo, se trata de un tema que genera gran cantidad de producción científica dando lugar a diversas revisiones bibliográficas sobre las tecnologías de inteligencia artificial y visión por computador en la industria alimenticia (Kakani et al., 2020) y sobre los distintos métodos de clasificación de frutas y verduras (Hameed et al., 2018). Por su parte, Di Wu y Sun (2013) revisaron trabajos sobre los fundamentos y aplicaciones de la visión por computador para la medición del color de los alimentos y analizaron las ventajas y desventajas de la visión artificial para la medición del color.

En lo relativo a aplicaciones para el aceite de oliva, Cano-Marchal et al. (2013) desarrollaron un sistema basado en visión artificial y reconocimiento de patrones para clasificar el contenido de impurezas insolubles de las muestras de aceite de oliva. Para ello, los autores probaron con tres vectores de entrada diferentes derivados del histograma de los canales de los espacios de color RGB, HSV y CIELAB, a los que se aplicó una extracción de características antes de la clasificación. Se emplearon varios métodos de extracción de características tanto lineales como no lineales y los clasificadores utilizados fueron las máquinas de soporte vectorial (Support Vector Machines o SVMs) y las redes artificiales de neuronas (Artificial Neural Networks o ANN). La mayor tasa de precisión se obtuvo usando extracción de características *Kernel Principal Component*

Analysis (KPCA) y una máquina de soporte vectorial con *kernel* polinómico de tercer grado.

En la misma línea, Gila et al. (2020) aplicó un sistema de visión artificial para estimar el contenido de humedad e impurezas en el aceite de oliva virgen. El valor de predicción obtenido con la visión artificial se comparó con el método de medición actual, obteniéndose una alta correlación positiva entre ambos y demostrando la utilidad de la visión artificial para la determinación de estos parámetros en el aceite de oliva virgen.

Furferi et al. (2007) elaboraron un software basado en la combinación de un enfoque de red neuronal artificial (ANN) y un método numérico para la estimación del nivel de acidez y del número de peróxidos del aceite de oliva producido en un proceso continuo de extracción. Para la estimación se realizaron mediciones de algunos parámetros agronómicos y tecnológicos comúnmente medidos por los técnicos que trabajan en las almazaras.

Martínez y Pontes (2015) propusieron un método basado en imágenes digitales y reconocimiento supervisado de patrones para la clasificación de muestras de aceite de oliva virgen extra entre varias marcas y la verificación de su adulteración con aceite de soja. Para ello, se empleó información del espacio de color RGB para desarrollar modelos de clasificación basados en análisis de discriminación lineal (Linear Discriminant Analysis o LDA) seleccionando las variables mediante el algoritmo de proyecciones sucesivas (Successive Projections Alogrithm o SPA) y un análisis discriminante de mínimos cuadrados parciales (PLS-DA).

Otros autores combinaron espectroscopia dieléctrica y visón por computador para evaluar las características del aceite de oliva durante su almacenamiento (Sanaeifar et al., 2018) o evaluar su oxidación (Sanaeifar y Jafari, 2019). Martínez Gila et al. (2018) estudiaron las relaciones entre el contenido en polifenoles y su respuesta en el espectro de luz infrarroja. Por otro

lado, Cano et al. (2021) emplearon una e-nose de uso general, en condiciones de laboratorio, para predecir el nivel de aroma afrutado y la presencia de defectos en aceite de oliva virgen, obteniendo un error de validación medio de 0,5 unidades para la predicción de frutado aroma mediante regresión de lazo; y 88% de precisión para la detección de defectos mediante regresión logística.

Además de estos trabajos, también se encuentran revisiones bibliográficas como la de González-Fernández et al., 2019 en la que revisaron la literatura existente sobre el uso de métodos predictivos bioinformáticos basados en ANN en relación con la producción, procesamiento y caracterización del aceite de oliva. En lo referente a estudios destinados al control del proceso de elaboración del aceite de oliva, Marchal et al. (2011), realizaron una revisión bibliográfica mostrando la situación actual y las perspectivas de futuro del proceso de elaboración del aceite de oliva virgen. Por otra parte, Ortega et al. (2016) revisaron trabajos sobre las tecnologías de detección más importantes que se utilizan cada vez más en la industria del aceite de oliva para supervisar y controlar su proceso de producción. Kalogianni et al. (2019) realizaron una revisión de la tecnología actual del procesamiento de aceite de oliva y compararon las tecnologías disponibles para las operaciones involucradas. Además, para cada operación unitaria involucrada en el procesamiento del aceite de oliva, se discutió el efecto de las variables del proceso en los parámetros en la determinación de la calidad del aceite de oliva. Por otro lado, Parra-López et al. (2015) examinaron los factores que condicionan la implementación de los Sistemas de Gestión de la Calidad que cumplen los requisitos de la norma ISO 9001 en la industria del aceite de oliva de Andalucía, y si la implementación de la esta norma está asociada con mejores prácticas de producción y comercialización.

Por otro lado, múltiples de las aplicaciones de la visión artificial en este campo se han dedicado a la clasificación de las aceitunas en función de la presencia de defectos o a inferir diferentes propiedades de las mismas, como el

índice de maduración o contenido de aceite. Díaz et al. (2004) presentaron una comparación de tres algoritmos: mínimos cuadrados parciales (Partial least squares o PLS), distancia de Mahalanobis y redes neuronales artificiales (Artificial Neural Networks o ANN); para clasificar las aceitunas de mesa de acuerdo con sus defectos y concluyó que los mejores resultados se obtienen utilizando ANN, con una precisión de clasificación del 90%. Riquelme et al. (2008) platearon utilizar tres análisis discriminantes consecutivos para el mismo objetivo, obteniendo una tasa de clasificación del 75% en la fase de validación. Guzmán et al. (2013b) propusieron un método directo para determinar en línea la madurez de las aceitunas, empleando imágenes en espectro visible para determinar el índice de madurez e imágenes en espectro infrarrojo para detectar defectos. Posteriormente, evaluaron un método de clasificación en función de los defectos basado en sistema de visión infrarroja (Guzmán et al., 2013a).

En cuanto al índice de maduración de las aceitunas, Furferi et al. (2010) desarrollaron un método que integra un sistema de visión artificial con un mayor refinamiento de los resultados utilizando una red neuronal artificial (Artificial Neural Network o ANN). Esta ANN combinada el índice de maduración preliminar proporcionado por los algoritmos iniciales con parámetros químicos obtenidos de curvas históricas de la región de donde proceden las aceitunas. Por otro lado, Guzmán et al. (2015) evaluaron el índice de maduración de diferentes muestras de aceitunas se mediante análisis de imágenes obtenidas mediante visión artificial, en el que se utilizaron algoritmos de segmentación por color y operadores para detectar aristas.

Para la predicción del contenido de aceite de las aceitunas, Ram et al. (2010) construyeron dos modelos basados en regresiones lineales y redes neuronales (ANN). Los mejores resultados se obtuvieron utilizando ANN, con correlaciones lineales de 0,81 para aceitunas Souri y 0,87 para aceitunas Picual.

Concluyendo con las aplicaciones de la visión artificial para aceitunas, cabe mencionar los estudios realizados para la clasificación de aceitunas. Gatica et

al. (2011) presentaron un método reconocer el diámetro de las aceitunas a partir de imágenes del olivo. Aguilera Puerto et al. (2015) proporcionaron un sistema de inspección automático, basado en visión artificial, para clasificar automáticamente los diferentes lotes de aceitunas que entran en el proceso de molienda. Esta clasificación se basaba en la diferenciación entre aceitunas molidas y de árbol. También presentaron un sistema en línea de visión artificial que clasifica automáticamente lotes de aceitunas de acuerdo con su nivel de calidad en diferentes categorías (Aguilera Puerto et al., 2019).

Hay algunos trabajos relacionados con las características visuales del aceite de oliva que no aplican técnicas de visión artificial, pero que vale la pena mencionar. Moyano et al. (2008a, 2008b) estudiaron la relación entre los parámetros de color del aceite de oliva en diferentes espacios de color para sus índices de clorofilas y carotenoides usando un espectrofotómetro. A su vez, Gordillo et al. (2011) estudiaron la influencia de la turbidez grado sobre el color y el aspecto de las muestras de aceite de oliva virgen, utilizando un turbidímetro y un espectrofotómetro.

En cuanto a las aplicaciones de las técnicas de visión por computadora a otros productos líquidos en la industria alimentaria, Hepworth et al. (2004) emplearon la visión por computador para determinar el tamaño y la velocidad de las burbujas en la cerveza. También es importante destacar el trabajo de Rashvand y Akbarnia (2019) que evaluaron un sistema inteligente portátil para detectar la adulteración del aceite de sésamo empleando tecnología de visión artificial. En este trabajo, se diseñaron y evaluaron redes neuronales artificiales con una capa oculta con un número diferente de neuronas.

Finalmente, mencionar el trabajo de Minz et al. (2018) en el que propusieron un método que aprovecha la calibración de los canales del espacio de color CIELAB mediante análisis de regresión múltiple; y el de Song et al. (2020) en que se presentó un novedoso sistema de sensores para la autenticación de alimentos, basado en la visión artificial y el reconocimiento de

patrones, usando un smartphone para generar una secuencia de luz con diferentes colores para iluminar las muestras y la cámara del dispositivo para recibir la luz reflejada a través de la grabación de un video.

1.3 Objetivos

El objetivo de este Trabajo Fin de Máster (TFM) es el desarrollo e implementación de un sistema de visión por computador para determinar el nivel de impurezas presentes en muestras de aceite de oliva virgen en condiciones de laboratorio.

Los objetivos específicos fueron los siguientes:

- Estudio previo de las imágenes y selección de características.
- Diseño de varios clasificadores mediante el ajuste de hiperparámetros.
- Evaluación de los distritos clasificadores y vectores de características.
- Selección del clasificador óptimo.
- Diseño e implementación de una Interfaz Gráfica de Usuario para el sistema.

1.4 Proceso de desarrollo

Para alcanzar el objetivo propuesto, el trabajo se desarrolló en cuatro fases. La primera fase consistió en la búsqueda de información sobre los distintos métodos de clasificación que finalmente se emplearon. En concreto, sobre los métodos de extracción de características, la búsqueda de rejilla (*Grid Search*), las funciones de la biblioteca Open CV y la de segmentación de imágenes.

En la segunda fase, se realizó una revisión bibliográfica de trabajos sobre sistemas para la detección de impurezas en el aceite de oliva virgen con el fin de identificar los aspectos tratados y aquellos que necesitaran mayores desarrollos,

diseñar este estudio experimental y, finalmente, documentar y contextualizar los resultados que se obtuvieran.

Tras analizar los trabajos obtenidos en la fase anterior, en la tercera fase se diseñaron varios sistemas capaces de cumplir con el objetivo establecido, para después evaluarlos y compararlos en la cuarta fase. Además, una vez seleccionado el sistema definitivo, se diseñó e implementó una interfaz gráfica de usuario para este sistema.

Por último, en la cuarta fase se evaluó el sistema desarrollado, comparando los resultados obtenidos con los de trabajos similares.

1.5 Estructura de la memoria

Esta memoria de TFM se estructura en ocho capítulos. En el primer capítulo, en el cual se enmarca este apartado, se realiza una introducción al tema desarrollado y se ofrece una descripción del objetivo principal y el proceso de desarrollo.

En el segundo capítulo se explican los conocimientos teóricos necesarios para la realización y compresión de este trabajo. Se recogen los fundamentos básicos de la inteligencia artificial y los problemas de clasificación. Además, se describen las herramientas de software empleadas en el desarrollo del trabajo.

En los capítulos siguientes, se detallan los materiales y métodos empleados en la realización de este trabajo, así como algunos conocimientos teóricos necesarios para su desarrollo y compresión.

En el tercer capítulo se expone el método empleado para la obtención de imágenes. A continuación, en el cuarto capítulo se describen los métodos empleados para la elaboración del vector de características.

En el quinto capítulo se explican con mayor profundidad los conocimientos teóricos necesarios para el diseño y comprensión de los distintos clasificadores comparados en este trabajo, y se detallan los métodos utilizados.

En el sexto capítulo se exponen los resultados obtenidos en apartados anteriores, y se selecciona el sistema definitivo. En el séptimo capítulo se diseña una interfaz gráfica de usuario que implemente el sistema seleccionado.

Por último, en el octavo capítulo se discuten los resultados y se detallan las conclusiones obtenidas en la realización del trabajo.

Adicionalmente, se incluye la bibliografía consultada y utilizada, así como tres anexos donde se adjunta el código desarrollado.

2 Fundamentos teóricos y herramientas de software

2.1 Inteligencia artificial y aprendizaje automático

La inteligencia artificial (AI, del inglés *Artificial Intelligence*) es una rama de la informática que consiste en la simulación de procesos de inteligencia humana por parte de máquinas, esencialmente sistemas informáticos. Estos procesos incluyen la adquisición de información y reglas para su uso (aprendizaje), el uso de reglas para llegar a conclusiones aproximadas o definitivas (razonamiento) y la autocorrección (Investigación y Ciencia, 2021).

El *Machine Learning* o aprendizaje automático es una aplicación de la AI que se centra en el desarrollo de programas informáticos capaces de generalizar comportamientos a partir de los datos recibidos, modificando su comportamiento en función de los datos de entrada (Contreras, 2016). Para poder aprender, el sistema debe tener la capacidad de deducir la función subyacente a los datos proporcionados.

Para un conjunto de datos de entrada, el programa aprende a asociar combinaciones de características distintivas a un elemento concreto. Una vez construido el modelo, se prueba para tartar de identificar dicho elemento en un conjunto de datos que no ha visto antes. El programa medirá el éxito de obtenido y utilizará esa información para reajustar el modelo, de forma que la próxima vez obtendrá mejores resultados. Este proceso iterativo continúa hasta que se obtenga un modelo con un elevado nivel de precisión (García Moreno, 2021).

El aprendizaje automático puede clasificarse en aprendizaje supervisado y aprendizaje no supervisado, dependiendo de la cantidad de información suministrada en el proceso de aprendizaje.

El aprendizaje supervisado es aquel en el que, para un conjunto de datos de datos de entrada, se conocen previamente los datos de salida (Contreras,

2016). Este tipo de aprendizaje consiste en aprender la relación entre dos conjuntos de datos: los datos de muestra *X* y una variable externa *y* que se trata de predecir, normalmente denominada objetivo o etiqueta de clase (Scikit-learn developers, 2020d). Consta de dos fases: la fase de entrenamiento, en la que se cuenta con un conjunto de datos para ajustar el algoritmo; y la fase de validación en la que utilizan datos nuevos para evaluar el rendimiento de la predicción.

Dentro del aprendizaje supervisado se encuentran los procedimientos de clasificación y regresión. Si el objetivo de la predicción es clasificar las observaciones en un conjunto de etiquetas de clase, en otras palabras, "nombrar" los objetos observados, se dice que la tarea es una tarea de clasificación. Sin embargo, si el objetivo es predecir una variable objetivo continua, se dice que es una tarea de regresión (Scikit-learn developers, 2020d).

Por otro lado, en el aprendizaje no supervisado no se conocen de antemano los datos de salida para un conjunto dado de datos de entrada. Intenta deducir relaciones buscando similitudes en el conjunto de datos de entrenamiento. Se utiliza para obtener una agrupación coherente de los datos en función de las relaciones entre las variables definidas en los datos. Un ejemplo de este tipo de aprendizaje es el método de agrupación (clusterizado) (Contreras, 2016).

2.2 Método de clasificación

El método de clasificación se utiliza para predecir un resultado con valor discreto dadas unas características. El método más simple de clasificación es el de clasificación binaria, donde se clasifica un registro de variables de entrada en 1 o 0. La clasificación múltiple es una extensión de la clasificación binaria (Contreras, 2016).

El problema de clasificación puede restringirse a considerar un problema de dos clases sin perder generalidad. En el problema de clasificación binaria, el objetivo es separar las dos clases mediante una función inducida por los datos

disponibles. La finalidad es producir un clasificador que trabaje bien en los ejemplos no vistos durante la fase de entrenamiento (Gunn, 1998).

Aprender los parámetros de una función de predicción y probarlos con los mismos datos es un error metodológico: un modelo que simplemente repita las etiquetas de las muestras que acaba de ver tendría una puntuación perfecta, pero no sería útil prediciendo datos que no se han mostrado aún (Scikit-learn developers, 2011).

Cuando el modelo seleccionado mapea casi perfectamente la tendencia de los datos de entrenamiento, pero falla en la generalización de nuevos registros, se considera *overfitting* (sobreajuste) (Contreras, 2016). Para evitar el sobreajuste, se dividen los datos disponibles en dos conjuntos diferentes: un conjunto de entrenamiento que se utiliza para aprender los parámetros del modelo predictivo y un conjunto de validación utilizado para evaluar el modelo predictivo ajustado (Scikit-learn developers, 2011).

Sin embargo, al definir estos dos conjuntos, reducimos drásticamente el número de muestras que pueden usarse para entrenar el modelo, y los resultados pueden depender de una elección aleatoria particular para el par de conjuntos (entrenamiento/validación) (Scikit-learn developers, 2011).

Una solución es dividir todos los datos varias veces consecutivas en diferentes conjuntos de entrenamiento y validación, y devolver el valor promedio de las puntuaciones de predicción obtenidas con los diferentes conjuntos. Este procedimiento se denomina validación cruzada. Este enfoque puede ser computacionalmente costoso, pero no desperdicia demasiados datos (como ocurre fijando un conjunto de prueba arbitrario), lo cual es una gran ventaja en problemas donde el número de muestras es muy pequeño (Scikit-learn developers, 2011).

La mayoría de los iteradores de validación cruzada admiten la generación de máscaras booleanas o índices enteros para seleccionar las muestras de una iteración determinada. El más común y el empleado en este trabajo es el iterador K-fold, que divide todas las muestras en *K* grupos (*folds*) de igual tamaño, siempre que sea posible. La función de predicción se entrena usando *K-1* grupos, y el grupo restante se usa en la validación. Si K es igual que el número de muestras, es equivalente a la estrategia de validación cruzada Leave-one-out (dejar uno fuera) (Scikit-learn developers, 2011).

Uno de los métodos de evaluación del rendimiento de un modelo de clasificación es la matriz de confusión. Esta matriz compara los valores reales con los predichos por el modelo de aprendizaje, lo que ayuda a comprobar el funcionamiento del modelo (Acevedo, 2020).

De la matriz de confusión pueden derivarse varios índices de medida. La tasa de error (*Error rate*) y la precisión o tasa de acierto (*Accuracy*) son las medidas más comunes e intuitivas derivadas de la matriz de confusión. La tasa de error (ERR) se calcula como el número de todas las predicciones incorrectas dividido por el número total de muestras del conjunto de datos. La mejor tasa de error es 0.0, mientras que la peor es 1.0 (Classifier evaluation with imbalanced datasets, 2015).

$$\mathrm{ERR} = \frac{\text{Número de clasificaciones erróneas}}{\text{Número total de clasificaciones}} \quad (1)$$

Por otro lado, la precisión (ACC) se calcula como el número de todas las predicciones correctas dividido por el número total de muestras del conjunto de datos. La mejor precisión es 1.0, mientras que la peor es 0.0 (Classifier evaluation with imbalanced datasets, 2015).

$$\mathrm{ACC} = \frac{\text{Número de clasificaciones correctas}}{\text{Número total de clasificaciones}} = 1 - \mathrm{ERR} \quad (2)$$

La precisión de validación cruzada se calcula como la media de la tasa de acierto (*accuracy*) obtenida para cada uno de los K diferentes conjuntos de entrenamiento.

Algunos de los clasificadores más empleados son las máquinas de soporte vectorial (*Support Vector Machines* o SVM), las redes neuronales artificiales (*Artificial Neural Networks* o ANN), los árboles de decisión, la regresión logística, el clasificador del vecino más cercano (K-Nearest Neighbour) o el clasificador de Bayes.

2.3 Herramientas de software

2.3.1 Lenguaje de programación. Python

Python es el lenguaje de programación empleado en el desarrollo de este Trabajo Fin de Máster.

Python es un lenguaje de programación que permite trabajar más rápido e integrar los sistemas de forma más efectiva (Python.org, 2021). Debido a su simplicidad, es uno de los lenguajes de programación más potentes que hay actualmente (Lenguajes de programación, 2020).

A continuación, se indican algunas de las características de este lenguaje de programación (Python.org, 2021):

- Es un lenguaje de propósito general, lo que significa que no está orientado a un fin concreto.
- Permite varios paradigmas de programación. Aunque su fuerte es la programación orientada a objetos existen otros estilos de programación para sus usuarios, como es la programación imperativa (con sentencias de bucle) o la programación funcional (con módulos y funciones).

- Es un lenguaje interpretado. Al programar en Python, no se compila el código fuente a código máquina, sino que hay un intérprete que ejecutará el programa basándose directamente en el código.
- Puede usarse en muchos dispositivos y sistemas operativos, ya que se han creado intérpretes para Unix, Linux, Windows y sistemas Mac Os.
- Es de tipado dinámico, lo que significa que cuando se declara una variable, no es necesario decirle de qué tipos son los datos. La variable se adapta a lo que escribimos cuando se ejecuta el programa.
- Es un software de libre distribución.
- Existen una gran cantidad de librerías y funciones que pueden utilizarse gracias a su extensa biblioteca. También tiene soporte para múltiples bases de datos.

En los siguientes subapartados se señalan las diferentes librerías de Python empleadas en la realización de este trabajo.

2.3.2 OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de software libre de visión artificial y aprendizaje automático (*Machine Learning*). Tiene soporte para los lenguajes C ++, Python, Java y MATLAB, y es compatible con Windows, Linux, Android y Mac OS. Al ser un producto con licencia BSD, OpenCV facilita que las empresas utilicen y modifiquen el código.

Se creó para proporcionar una infraestructura común para aplicaciones de visión por computador y para acelerar el uso de la percepción automática (*Machine Perception*) en los productos comerciales.

OpenCV se inclina principalmente hacia aplicaciones de visión en tiempo real. Sus algoritmos pueden usarse para análisis y procesamiento de imágenes

o vídeos, detección y reconocimientos de rostros, identificación y seguimiento de objetos, clasificación acciones humanas en videos, reconstrucción 3D, producir nubes de puntos 3D a partir de cámaras estereoscópicas, reconocimiento de placas de vehículos, análisis de formas, realidad aumentada, búsqueda de imágenes similares de una base de datos de imágenes, entre muchas otras aplicaciones (OpenCV, 2020).

2.3.3 Scikit-Learn

Scikit-learn es una biblioteca de código abierto de uso general para el análisis de datos escrita en Python. Se basa en otras bibliotecas de Python: NumPy, SciPy y matplotlib (RIP Tutorial, 2021) y puede emplearse comercialmente por su licencia BSD.

Scikit-learn contiene implementaciones para diferentes algoritmos de aprendizaje automático y admite tanto aprendizaje supervisado como no supervisado. También proporciona varias herramientas para el ajuste de modelos, preprocesamiento de datos, selección y evaluación de modelos, entre otras (Scikit-learn developers, 2020c).

A continuación, se describen brevemente el ajuste y la predicción de estimadores, los pasos de preprocesamiento, las canalizaciones, las herramientas de validación cruzada y las búsquedas automáticas de hiperparámetros (Scikit-learn developers, 2020c).

Proporciona docenas de algoritmos y modelos de aprendizaje automático integrados, llamados estimadores. Cada estimador se puede ajustar a algunos datos utilizando su método de ajuste, *fit*. Este método generalmente admite dos entradas:

- La matriz de muestras *X*, en la que las muestras se representan como filas y las características se representan como columnas.

- Los valores objetivo *y*, que son números reales para tareas de regresión, o enteros para clasificación (o cualquier otro conjunto de valores discretos). Para las tareas de aprendizaje no supervisadas, no es necesario especificar *y*. Suele ser una matriz de una dimensión en la que cada fila contiene el valor objetivo de la muestra en fila correspondiente de la matriz *X*.

Por lo general, se espera que tanto *X* como *y* sean *arrays* de *Numpy* o tipos de datos equivalentes, aunque algunos estimadores funcionan con otros formatos, como matrices dispersas (Scikit-learn developers, 2020c).

Una vez ajustado el estimador, puede utilizarse para predecir los valores objetivo de nuevos datos, sin que sea necesario volver a entrenarlo (Scikit-learn developers, 2020c).

A menudo, el flujo de trabajo del aprendizaje automático se compone de diferentes partes. Una secuencia típica consta de un paso de preprocesamiento que transforma o introduce los datos y un estimador final que predice los valores objetivo (Scikit-learn developers, 2020c).

En Scikit-learn, los modelos de preprocesamiento y transformación siguen la misma interfaz de programación de aplicaciones (API) que los estimadores. Los modelos de transformación, en lugar de un método de predicción tienen un método de transformación que devuelve la matriz de muestras *X* transformada. Tanto los estimadores como los transformadores pueden combinarse en un único objeto (*Pipeline*) que ofrece la misma API que el de un estimador normal: puede tanto ajustarse y como utilizarse para predecir con los métodos *fit* y *predict* (Scikit-learn developers, 2020c).

Sin embargo, ajustar un modelo con algunos datos no implica que este predecirá bien sobre datos desconocidos. Por esta razón, Scikit-learn

proporciona herramientas para la evaluación de modelos, en particular para la validación cruzada (Scikit-learn developers, 2020c).

Por otro lado, los estimadores tienen parámetros (a menudo llamados hiperparámetros en la literatura) que se pueden ajustar. El poder de generalización de un estimador a menudo depende de unos pocos parámetros. Scikit-learn proporciona herramientas para encontrar automáticamente las mejores combinaciones de parámetros mediante validación cruzada (Scikit-learn developers, 2020c).

En los siguientes apartados se especifican los modelos empleados en las fases de preprocesamiento de datos y estimación de resultados.

2.3.4 NumPy

Numpy es la biblioteca principal para la ciencia computacional en Python. Proporciona un objeto de matriz multidimensional de alto rendimiento (*array*) y herramientas para trabajar con estas matrices (Johnson, 2021). Distribuida bajo una licencia BSD, NumPy es desarrollado y mantenido públicamente en GitHub, a través del consenso de NumPy y de la comunidad científica Python en general. Es compatible con una amplia gama de plataformas informáticas y de hardware, y funciona bien con bibliotecas distribuidas, de GPU y de arreglos dispersos.

Rápidos y versátiles, los conceptos de vectorización, indexación y transmisión de NumPy son los estándares de facto de la computación de arrays en la actualidad. Ofrece varias herramientas de cálculo numérico como funciones matemáticas exhaustivas, generadores de números aleatorios, rutinas de álgebra lineal y transformadas de Fourier. Además, la sintaxis de alto nivel de NumPy lo hace accesible y productivo para programadores de cualquier origen o nivel de experiencia (*NumPy*, 2021).

2.3.5 Matplotlib

Matplotlib es una biblioteca de representación gráfica de Python 2D que produce figuras de calidad en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Puede utilizarse en scripts de Python, los shells de Python e IPython, Jupyter Notebooks, servidores de aplicaciones web y cuatro kits de herramientas de interfaz gráfica de usuario (Matplotlib development team, 2020).

Con esta biblioteca pueden generarse gráficos, histogramas, espectros de potencia, gráficos de barras, gráficos de error, gráficos de dispersión, etc., con pocas líneas de código. Para un trazado de gráficas simple, el módulo *pyplot* proporciona una interfaz similar a MATLAB, especialmente cuando se combina con IPython. Para usos más avanzados, proporciona control total de los estilos de línea, las propiedades de la fuente, las propiedades de los ejes, etc., a través de una interfaz orientada a objetos o mediante un conjunto de funciones similares a las de MATLAB (Matplotlib development team, 2020).

3 Obtención de imágenes

Las imágenes empleadas en este TFM fueron las realizadas por Cano Marchal, et al. (2013) para la investigación *Expert system based on computer vision to estimate the content of impurities in olive oil samples.*

El artículo recoge que, para que el proceso de medición fuese constante durante todas las pruebas, se verificaron diferentes aspectos antes de adquirir la información. Estos aspectos fueron: ausencia de polvo o suciedad, poca luz externa en comparación con el sistema de iluminación (reduciendo así el ruido proveniente de la luz envolvente), temperatura constante y ausencia de golpes o vibraciones del equipo (Cano Marchal, et al. 2013).

La figura 2 muestra el equipo experimental utilizado para la adquisición de imágenes.

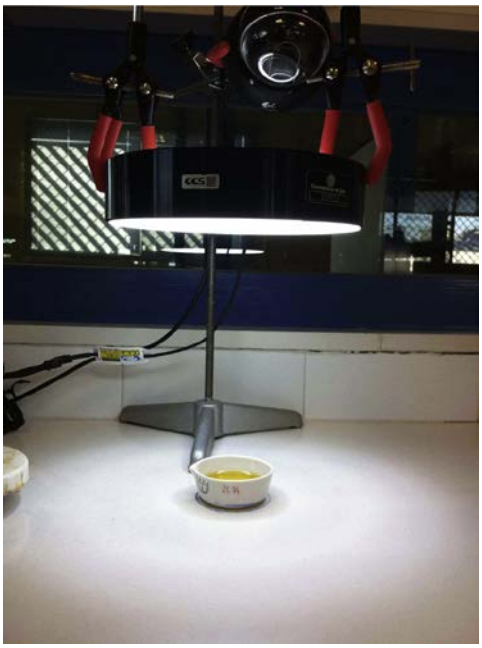


Figura 2. Configuración de adquisición de imágenes, compuesta por el anillo de iluminación y la cámara (parte superior). En la parte inferior de la imagen está el crisol de porcelana que contiene la muestra de aceite de oliva (Cano Marchal, et al. 2013).

El sistema estaba formado por una cámara web Logitech LZ836BP de resolución de 2 MP y un dispositivo de iluminación LED en forma de anillo. Esta fuente de iluminación tenía una potencia de 17 W, un diámetro de 18 cm y se colocó 25 cm por encima de las muestras. Las imágenes se realizaron mediante el software proporcionado con la cámara web con un ordenador de escritorio estándar Pentium IV.

El conjunto de muestras disponible estaba compuesto por 165 muestras, de las cuales 87 pertenecían a la primera clase, 54 a la segunda y 24 a la tercera. El desequilibrio entre la cantidad de muestras que componían cada clase se debe a la frecuencia con la que estos valores se dan naturalmente en el proceso de elaboración del aceite, ya que las muestras utilizadas las suministró un laboratorio especializado en análisis de aceite de oliva a partir de las que éste recibe para dicho análisis. También, se intentaron construir muestras artificiales para el mayor nivel de impurezas, pero las muestras obtenidas mostraron notorias diferencias con las naturales, por lo que fueron descartadas (Cano Marchal, et al. 2013).

Grado de impurezas	**Porcentaje de impurezas**	**Nº de muestras**	**Nº de muestras acumulado**
Bajo (i ≤ 0,04)	0,02	8	87
	0,03	79	
Medio (0,04 < i < 0,06)	0,04	41	54
	0,05	13	
Alto (i ≥ 0,06)	0,06	16	24
	0,07	6	
	0,1	2	

Tabla 1. Muestras de aceite perteneciente a cada clase (i denota el porcentaje de impurezas de la muestra)

4 Obtención del vector de características

La existencia de impurezas insolubles y la cantidad total de las mismas vienen determinadas por el color de los píxeles, independientemente de su posición. Por esta razón, los datos considerados como parámetros de entrada fueron los histogramas de los canales en tres espacios de color: RGB, CIELAB y HSV. Se consideraron estos tres espacios de color diferentes para construir un vector de características con varias combinaciones no lineales de los mismos datos originales (Cano Marchal, et al. 2013).

Con relación a lo anterior, es necesario destacar que un número elevado de características puede ser perjudicial a la hora de entrenar el modelo, haciendo que éste se amolde demasiado a las muestras proporcionadas y no sea capaz de predecir muestras nuevas. Esto es lo que se conoce como sobreajuste (*overfitting*). Una de las acciones llevadas a cabo para reducir el tamaño de los datos de entrada es agrupar los histogramas. Para obtener los histogramas agrupados de los distintos canales, se elaboraron dos funciones cuyo propósito y parámetros se recogen en los cuadros 1 y 2.

agrupar_hist (hist, n=2)

Agrupa el histograma hist en función del parámetro n.

- **hist:** *list*. Histograma de la imagen de muestra
- **n:** *int*. Parámetro de agrupación. Por defecto toma el valor 2.

Cuadro 1. Función auxiliar para la agrupación de histogramas

hist_agrupado (*img*, *esp = 'BGR'*, *canal = 0*, *n=2*)

Obtiene el histograma del canal requerido y lo agrupa empleando la función auxiliar agrupar_hist.

- **img:** *array of uint8.* Imagen de muestra en formato BGR
- **esp:** *string*. Espacio de color en el que calcular el histograma. Toma los valores 'BGR', 'HSV' o 'CIELAB'. Valor por defecto *esp = 'BGR'*
- **canal:** *string*. Canal en el que se calcula el histograma. Toma los valores 0, 1 o 2. Valor por defecto *canal = 0*
- **n:** *int*. Parámetro para la agrupación del histograma. Por defecto toma el valor 2

Cuadro 2. Función que obtiene el histograma agrupado de una imagen

4.1 Segmentación de imágenes

Con el fin extraer la región de interés, se realizó una segmentación de las imágenes. El objetivo de este procedimiento es el de compartimentar imágenes en regiones separadas. En este sentido, el color es una de las formas más sencillas de identificar qué píxeles pertenecen a cada objeto. De esta manera, asociando un rango de color a cada uno de los objetos esperados en la imagen, puede realizarse una operación de búsqueda que determine qué etiqueta de objeto debería asignarse a cada píxel de dicha imagen (Sangwine y Horne, 1998).

La forma más sencilla de segmentar por colores es establecer rangos de valores para las componentes del espacio de color RGB (Red – Green – Blue) pero, mientras que para ciertos colores es fácil determinar un rango de valores adecuado, este espacio de color no siempre es práctico para detectar colores. Por esta razón, es más adecuado el espacio de color HSV (*Hue – Saturation – Value*), que permite separar las cualidades que definen el color en tres canales (Gl4r3, 2020).

Una vez segmentada la región de interés, es importante eliminar el ruido de la imagen, para lo que se aplicaron transformaciones morfológicas de apertura y cierre. La operación de apertura permite eliminar el ruido que se encuentre fuera de la región segmentada, mientras que la de cierre sirve para limpiar el interior de la misma. Para realizar estas operaciones, es necesario definir un elemento estructurante (*kernel*). El resultado dependerá del tamaño y forma de este elemento (Tutor de Programación, 2017).

Tras varias pruebas con las imágenes de muestra, se definieron los valores máximos y mínimos para cada componente, mostrados en la tabla 2. En cuanto a las operaciones morfológicas, se asignó un *kernel* cuadrado de tamaño 7x7. En la figura 3 se observa la máscara de segmentación obtenida para una de las muestras con estos valores.

Canal	Valor máximo	Valor mínimo
H	0	40
S	210	255
V	50	255

Tabla 2. Valores máximos y mínimos para el rango de segmentación

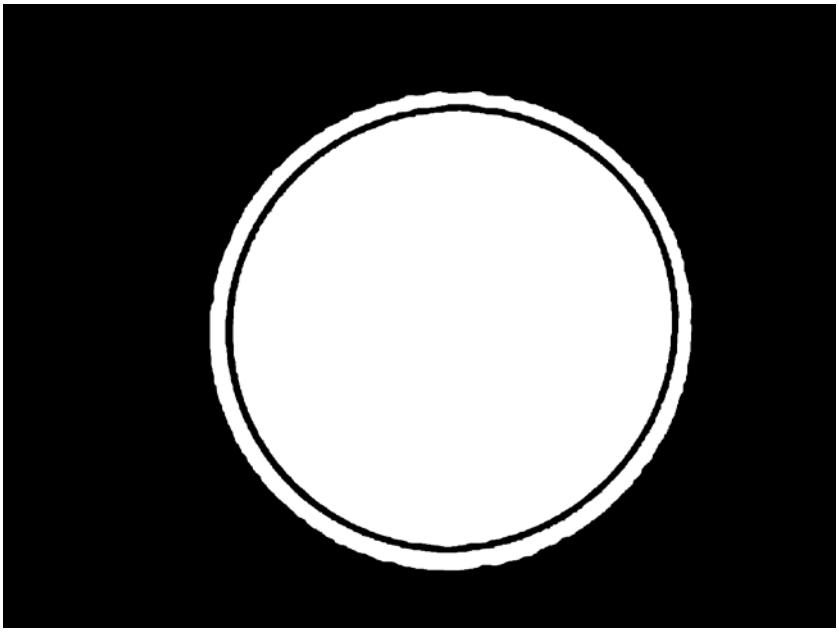


Figura 3. Máscara de segmentación de la muestra 10_10__0

Para realizar la segmentación de imágenes, se aplicó la función img_segmentation, cuya descripción y parámetros se pueden encontrar en el cuadro 3. Esta función convierte la imagen al formato HSV y crea una matriz cuadrada de ceros de dimensiones (*kernel_dim*, *kernel_dim*) como elemento estructurante para las operaciones morfológicas. A continuación, se obtiene una máscara con los valores que se encuentran dentro del rango utilizando la función cv2.inRange de la biblioteca OpenCv. Esta función incorporada toma como parámetros de entrada la imagen y los rangos superior e inferior de los valores a segmentar. Posteriormente devuelve una máscara binaria del tamaño de la imagen en la que los valores 1 indican los píxeles que están dentro del rango (Stone, 2018). Por último, se aplican las transformaciones morfológicas de apertura y cierre con la función cv2.morphologyEx de la biblioteca de OpenCv. La salida de la función expuesta en el cuadro 3 es una conjunción lógica bit a bit entre la imagen y la máscara que da como resultado la imagen segmentada.

img_segmentation (*img*, *low_range*, *high_range*, *kernel_dim*)

Segmenta una imagen y aplica operaciones morfológicas de apertura y cierre al resultado.

- **img:** *array of uint8.* Imagen de muestra en formato BGR
- **low_range:** *array*. Valores mínimos que pueden adoptar los canales H, S y V respectivamente
- **high_range:** *array*. Valores máximos que pueden adoptar los canales H, S y V respectivamente
- **kernel_dim:** *int*. Dimensión del elemento estructurante para las operaciones morfológicas

Cuadro 3. Función que segmenta una imagen dada y aplica una transformación morfológica de apertura y cierre

En la figura 4 puede observarse un ejemplo de imagen de muestra segmentada.

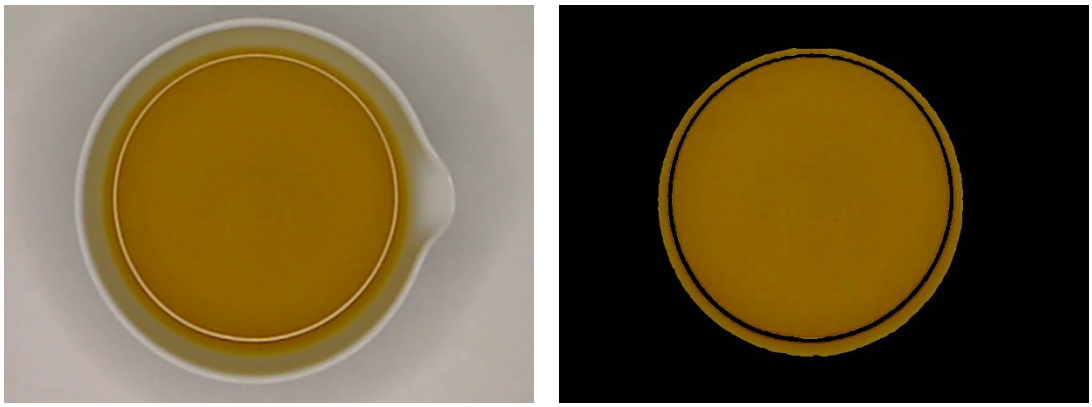


Figura 4. Imagen de muestra i0_03__28 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.03%

4.2 Composición del vector de características

Inicialmente, se consideró como vector de características el 'set 3' desarrollado por Cano Marchal, et al. (2013). En esta configuración solo se tuvieron en cuenta los canales *a* y *b* del espacio de color CIELAB y los canales *H* y *S* del espacio de color HSV. Este vector de características se construyó a partir de dos subconjuntos de parámetros. El primero se formó con los valores de cada elemento del canal del histograma en orden descendente. El segundo se definió con el índice que el elemento del primer subconjunto tenía en el histograma original (figura 5).

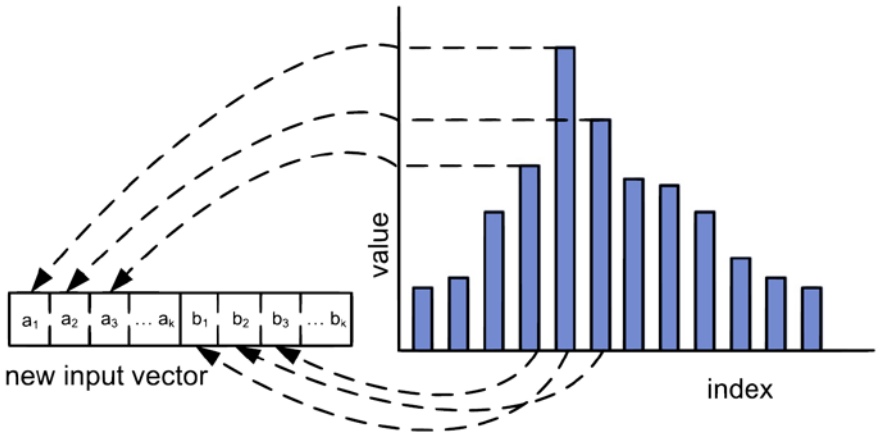


Figura 5. Construcción del vector de entrada final a partir del histograma original de un canal. El primer subconjunto del vector son los valores del histograma en orden descendente (Cano Marchal, et al. 2013).

Las pruebas iniciales no fueron del todo satisfactorias, por lo que se planteó el uso de otra configuración para formar el vector de características. Para comprobar cuáles eran los canales más adecuados se representaron los histogramas de cada canal de la imagen en los tres espacios de color. Así, se representaron los histogramas de las imágenes segmentadas y sin segmentar. Para obtener los histogramas de los distintos canales, se elaboraron dos funciones cuyo propósito y parámetros se recogen en los cuadros 4 y 5.

Histogramas (*n*, *n_H*, *n_Lab*)

Obtiene y representa gráficamente los histogramas agrupados de los canales de las imágenes de muestra para tres espacios de color: RGB, CIELAB y HSV.

- **n:** *int.* Parámetro de agrupación general
- **n_H:** *int.* Parámetro de agrupación del canal H
- **n_Lab:** *int.* Parámetro de agrupación de los canales del espacio de color CIELAB

Cuadro 4. Función que representa los histogramas agrupados de las imágenes de muestra para tres espacios de color

HistSegmentados (*n*, *n_H*, *n_Lab*)

Obtiene y representa gráficamente los histogramas agrupados de los canales de las imágenes de muestra segmentadas para tres espacios de color: RGB, CIELAB y HSV.

- **n:** *int.* Parámetro de agrupación general
- **n_H:** *int.* Parámetro de agrupación del canal H
- **n_Lab:** *int.* Parámetro de agrupación de los canales del espacio de color CIELAB

Cuadro 5. Función que representa los histogramas agrupados de los canales de las imágenes segmentadas para tres espacios de color

Estas dos funciones tienen como entrada tres parámetros de agrupación diferentes debido a que, en principio, no se dividirían todos los canales entre el mismo número por no tener los mismos elementos. De esta forma, podría elaborarse un vector de características en el que los histogramas de los canales empleados tuviesen una magnitud similar. En cuanto a la representación de los histogramas, estas funciones representan gráficamente los histogramas de los canales RGB, HSV y CIELAB agrupando las muestras por el contenido de impurezas insolubles (bajo, medio y alto). También representa en un mismo cuadro los histogramas de los canales que forman el primer vector de características descrito para que sea más fácil su comparación.

4.3 Datos extraídos

Para extraer los datos de las imágenes de muestra para cada una de las configuraciones del vector de características consideradas, se elaboró un programa que calculara el vector de características de cada imagen y almacenara los datos en una estructura de datos para facilitar su aplicación en apartados posteriores. A continuación, se expone el funcionamiento del programa que se recoge en el diagrama de flujo de la figura 6, similar en todas las configuraciones seleccionadas.

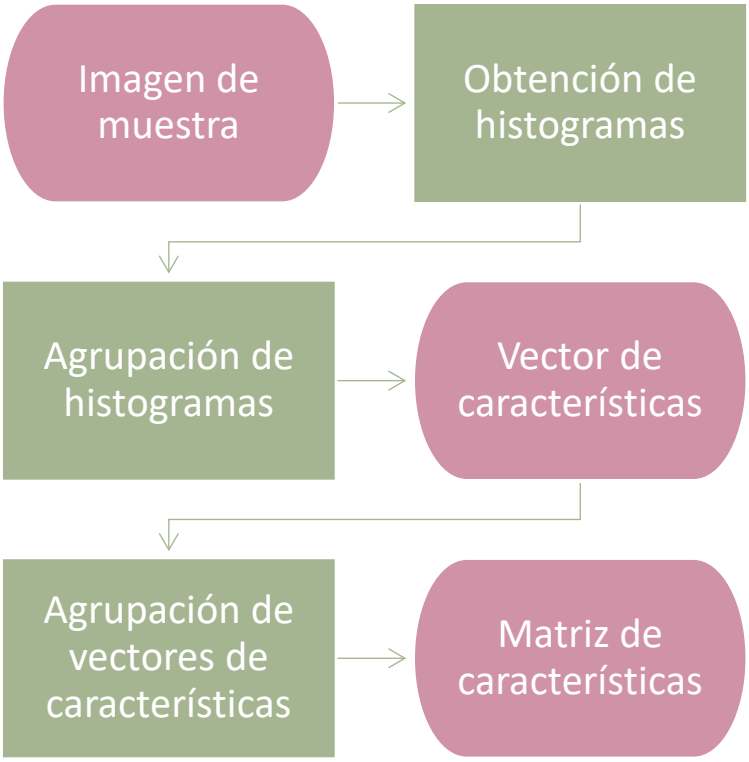


Figura 6. Esquema del proceso de obtención de la matriz de características

En primer lugar, el programa carga las imágenes de muestra y aplica la función *img_segmentation*, definida en el cuadro 3 del apartado anterior. Las imágenes segmentadas (variable *muestra*) se almacenan en la lista *images*. Para cada muestra, se obtiene mediante la función expuesta en el cuadro 2 el histograma agrupado de los canales que componen el vector de características. Estos histogramas, en formato lista, se juntan en una sola variable que se convierte a tipo *array* (elemento de la biblioteca Numpy). Posteriormente, los vectores resultantes para cada una de las imágenes se adjuntan a la matriz *p_vector* (elemento tipo *array* de Numpy). Esta es la matriz de características, en la que cada fila es el vector de características de una muestra y cada columna es una característica (correspondiente a cada elemento de los histogramas).

Una vez obtenida la matriz de características, se almacenan tanto la matriz como los datos de las muestras en un objeto de Python similar al diccionario (*bunch*) que facilita el manejo de conjuntos de datos. Los elementos de este objeto son:

- **data (array of int32):** matriz de características
- **target (array of int32):** grado de impurezas correspondiente a cada muestra
- **target_names (list):** nombres de los grados de impurezas (bajo, medio o alto)
- **target_level (array of int32):** nivel de impurezas correspondiente a cada muestra
- **level_names (list):** porcentajes de correspondiente al nivel impureza (str)
- **features (list):** canal al que pertenece cada elemento del vector de características
- **features_names (list):** nombres de los canales empleados

En el caso del vector de características compuesto por dos subconjuntos de datos (Cano Marchal, et al. 2013), se requirió una función adicional que, una vez obtenidos los histogramas agrupados, reordenase sus elementos para obtener la configuración deseada (figura 5). La función que se encarga de esta reordenación de datos es la que parece en el cuadro 6, que se emplea en lugar de la hist_agrupado (cuadro 2). En la figura 7 se muestra un esquema del proceso.

param_vector (*img*, *esp = 'BGR'*, *canal = 0*, *n=2*)

Esta función devuelve una lista con el vector de características de la imagen dada para un espacio de color y canal especificados. Utiliza las funciones agrupar_hist e hist_agrupado y sus parámetros de entrada coinciden con los de esta última. El vector de características se construye a partir de dos subconjuntos de parámetros: el primero se define con los valores de cada elemento del canal del histograma en orden descendente; el segundo se define con el índice que el elemento del primer subconjunto tenía en el histograma original

- **img:** imagen de muestra en formato BGR
- **esp:** *string*. Espacio de color en el que calcular el histograma. Toma los valores 'BGR', 'HSV' o 'CIELAB'. Valor por defecto *esp = 'BGR'*
- **canal:** *string*. Canal en el que se calcula el histograma. Toma los valores 0, 1 o 2. Valor por defecto *canal = 0*
- **n:** *int*. Parámetro para la agrupación del histograma. Por defecto toma el valor 2

Cuadro 6. Función que calcula el vector de características siguiendo la configuración propuesta por Cano Marchal, et al. (2013)

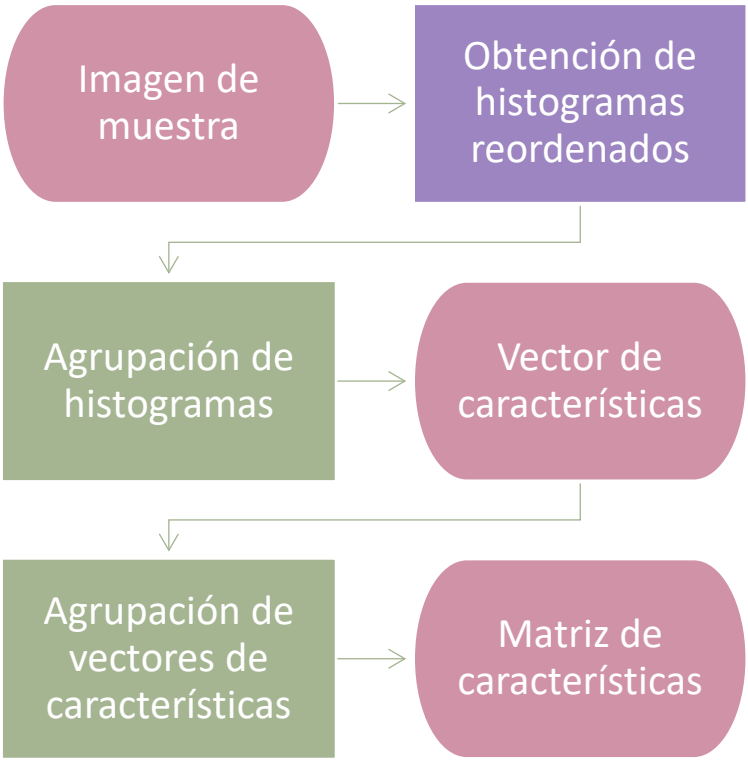


Figura 7. Esquema del proceso de obtención de la matriz de características siguiendo la configuración propuesta por Cano Marchal, et al. (2013)

4.4 Normalización

Los datos extraídos inicialmente, fueron sometidos a varias técnicas de preprocesamiento para prepararlos de cara a las a las siguientes etapas del proceso. La primera fase de este preprocesamiento fue la normalización, una técnica común de escalado de características en *machine learning* (aprendizaje automático).

El algoritmo de *machine learning* funciona con números, de modo que, si hay una diferencia de rango significativa, los números de mayor rango tendrán un papel más decisivo en el entrenamiento del modelo. Es por esto por lo que el escalado de características es uno de los pasos más importantes del preprocesamiento de datos (Roy, 2020).

Para normalizar los datos de entrada, se empleó la función MinMaxScaler de Scikit-learn. Este escalador transforma las características escalándolas a un rango dado, en este caso (0,1).

4.5 Extracción de características

Tras la normalización de los datos se llevó a cabo una extracción de características. Esta técnica tiene como objetivo reducir el número de características de un conjunto de datos, creando a partir de las características existentes un nuevo conjunto que debe poder resumir la mayor parte de la información contenida en el conjunto original (Ippolito, 2019).

El proceso de extracción de características no solo permite reducir la cantidad de recursos necesarios para el procesamiento; también puede reducir la cantidad de datos redundantes para un análisis determinado. Además, la reducción de los datos de entrada y de los esfuerzos de la máquina para crear combinaciones de variables (características) facilitan la velocidad de aprendizaje y generalización del proceso de *machine learning (*Ashwin, 2020).

De esta forma, se empleó un análisis de componentes principales o PCA (*Principal Component Analysis*). La base de esta técnica es encontrar las direcciones de máxima variabilidad, de modo que las *k* primeras proyecciones conserven el máximo de información posible que permita reconstruir los datos con error cuadrático mínimo (Müller et al., 2001). Los datos originales se proyectan en un conjunto de ejes ortogonales y cada eje se clasifica de acuerdo con el módulo del autovalor correspondiente (Ippolito, 2019).

Por otra parte, según Cano Marchal, et al. (2013), al ser un algoritmo de aprendizaje no supervisado, el PCA no implica ningún riesgo de sobreajuste. Sin embargo, no garantiza que las proyecciones de los datos en las direcciones obtenidas sean las más discriminatorias entre clases. Además, es posible perder la información discriminante si su variabilidad es baja y el número de componentes seleccionados no es lo suficientemente alto.

4.5.1 Visualización de la separación entre clases

Para visualizar la separación entre las clases tras aplicar el método de extracción de características PCA, se empleó el módulo PairPlot en las cuatro configuraciones seleccionadas del vector de características. Este módulo de la biblioteca *seaborn* traza relaciones por pares en un conjunto de datos creando una cuadrícula de forma que cada componente se compartirá en el eje *y* a lo largo de una fila y en el eje *x* a lo largo de una columna. Los gráficos de la diagonal muestran una estimación de la densidad del núcleo en capas (KDE) (Waskom, 2021).

Al visualizar la relación entre las componentes creadas por la técnica de extracción de características pudo comprobarse si el nuevo conjunto de característica permitía una buena separación entre clases. La visualización de las seis componentes de PCA calculadas para todo el conjunto de características con el vector de características HSab Inicial se muestra en la figura 8. Los

resultados obtenidos para el resto de los vectores de características se exponen en el apartado de resultados.

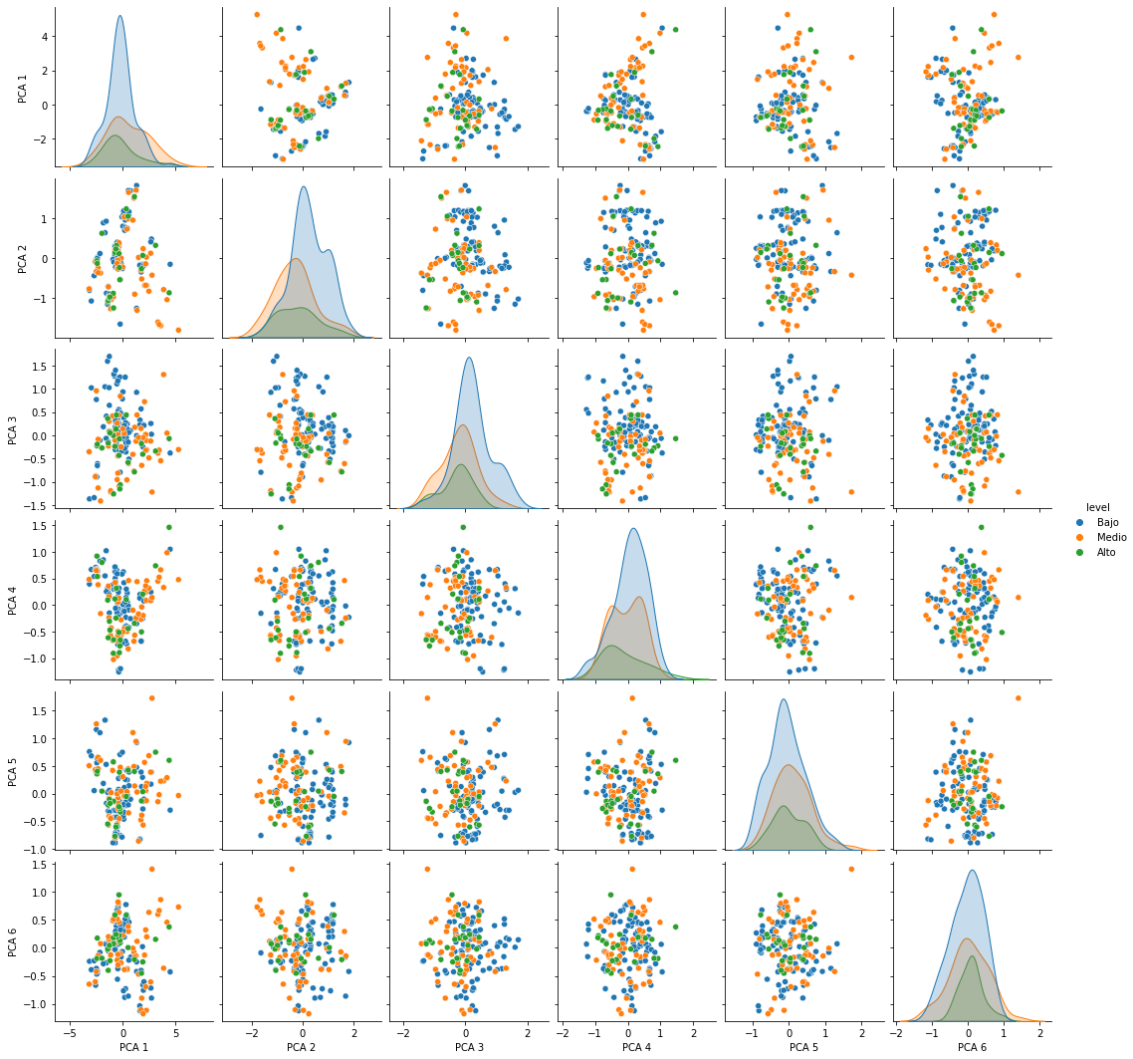


Figura 8. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características HSab Inicial.

5 Obtención del clasificador

Una vez calculados los distintos vectores de características, se ajustaron los hiperparámetros de varios clasificadores. Posteriormente, se validaron los clasificadores óptimos obtenidos para encontrar el que mejor determinase el nivel de impurezas contenido en las muestras de aceite.

A continuación, se exponen los procesos de diseño y validación de los clasificadores evaluados.

5.1 Diseño del clasificador

En el diseño del clasificador se consideraron las máquinas de soporte vectorial (*Support Vector Machines* o SVMs), un clasificador basado en los árboles de decisión (*Random Forest*) y la regresión logística multinomial.

El enfoque de diseño fue común para todos los métodos de clasificación considerados. Para ajustar los hiperparámetros del clasificador, se realizó una búsqueda de rejilla con los valores de cada hiperparámetro, empleando el conjunto completo de características y una validación cruzada K-Fold. Los datos de entrenamiento fueron los únicos utilizados para ajustar los métodos de normalización y extracción de características. Para la búsqueda de rejilla se ejecutó la clase GridSearchCV de Scikit-learn. Este es un objeto que, dados unos datos, calcula la puntuación durante el ajuste de un estimador para una cuadrícula de parámetros y elige aquellos que maximizan la puntuación máxima de validación (Scikit-learn developers, 2020d).

Una vez obtenidos los valores de los parámetros en cada iteración, se estudió el funcionamiento del clasificador calculando su precisión (*accuracy*) con los datos de validación, a los que se les aplicaron los métodos de normalización y extracción de características previamente ajustados con los datos de entrenamiento. Al separar los datos en conjuntos de entrenamiento y validación

antes de aplicar la búsqueda de hiperparámetros se obtuvieron *k* clasificadores que se compararon entre sí por su puntuación media. De entre estos valores se eligió el clasificador óptimo (uno por cada algoritmo de clasificación evaluado).

En el caso del clasificador de árbol de decisión (*Random Forest*), el proceso de diseño se dividió en dos partes. Se realizó un primer ajuste de los hiperparámetros empleando una búsqueda aleatoria en una cuadrícula de parámetros más amplia. Después, se llevó a cabo una búsqueda de rejilla en torno a los mejores valores obtenidos en la prueba anterior para determinar los parámetros del clasificador. Esto se debe a que el número de hiperparámetros ajustados para este clasificador y los valores asignados daban lugar a una cuadrícula mucho mayor que en el resto de los clasificadores y el coste computacional habría sido muy elevado.

Para la búsqueda inicial de los hiperparámetros del clasificador se ejecutó RandomizedSearchCV de Scikit-learn. De esta manera, pudo definirse una cuadrícula de valores de hiperparámetros y muestrearla aleatoriamente, haciendo una validación cruzada para cada combinación de valores. La ventaja principal de una búsqueda aleatoria es que no se prueba cada combinación, sino que el algoritmo selecciona al azar una combinación de características para cada iteración, permitiendo muestrear una amplia gama de valores (Koehrsen, 2018).

Tanto GridSearchCV como RandomizedSearchCV emplean la validación cruzada para evaluar el rendimiento del modelo, de modo que cada combinación de valores de hiperparámetros aporta varios cálculos diferentes que se promedian para obtener un comportamiento medio del clasificador en el espacio de parámetros (Sawtelle, 2016).

Como muestra la figura 9, se realizan dos bucles de validación cruzada de forma paralela: uno realizado por el estimador GridSearchCV para obtener los hiperparámetros y el otro empleado para medir la predicción del estimador. Los

resultados obtenidos son estimaciones no sesgadas de la puntuación de predicción sobre nuevos datos (Scikit-learn developers, 2020d).

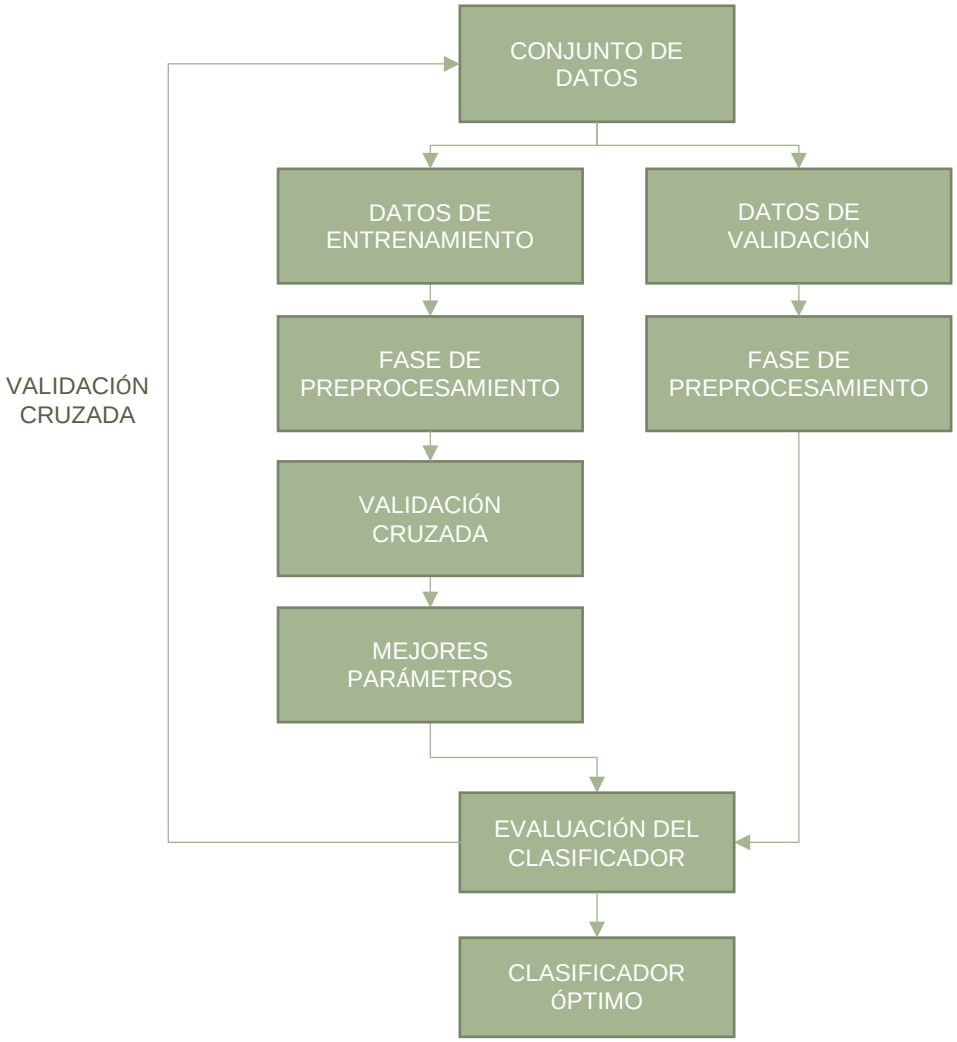


Figura 9. Esquema del diseño de clasificadores mediante validación cruzada anidada.

En los siguientes subapartados, se desarrollan los fundamentos teóricos necesarios para comprender el funcionamiento de los distintos algoritmos de clasificación evaluados.

5.1.1 Máquinas de soporte vectorial

Las máquinas de soporte vectorial (*Support Vector Machines* o SVM) son un conjunto de métodos de aprendizaje supervisados utilizados para la

clasificación, regresión y detección de valores atípicos (Scikit-learn developers, 2020e). Son modelos discriminantes, es decir, tratan de encontrar una combinación de muestras para construir un plano maximizando el margen entre las dos clases (Scikit-learn developers, 2020d).

En su forma más sencilla, resuelven el problema de clasificación entre dos clases linealmente separadas encontrando el hiperparámetro que maximiza el margen entre ambas clases (Burges, 1998), pero también pueden extenderse a problemas de clasificación multiclase usando diferentes enfoques, como el "uno contra uno" (*one-vs-one*) o el "uno contra todos" (*one-vs-rest)*. Además, pueden generalizarse para los casos no separables linealmente, relajando las restricciones mediante variables de holgura y un término de coste adicional, así como permitiendo el uso del *kernel* para mapear los datos en un espacio de alta dimensión y buscar el hiperplano de separación (Cano Marchal, et al. 2013).

La regularización de las SVM se establece mediante el parámetro C. De esta forma, un valor pequeño de C significa que el margen se calcula utilizando muchas o todas las muestras alrededor de la línea de separación (mayor normalización). Un valor alto de C significa que el margen se calcula en observaciones cercanas a la línea de separación (menor normalización) (Scikit-learn developers, 2020d).

Por otra parte, las SVM son efectivas en espacios de grandes dimensiones, incluso cuando el número de características es mayor al número de muestras. También son eficientes en la memoria, ya que utilizan un subconjunto de puntos de entrenamiento en la función de decisión (vectores de soporte). Además, son un método versátil, ya que pueden especificarse diferentes *kernels* para la función de decisión (Scikit-learn developers, 2020e).

Cuando el número de características es mucho mayor que el número de muestras es crucial seleccionar un *kernel* y un término de regularización para evitar el *overfitting* (Scikit-learn developers, 2020e). Para este trabajo, se

evaluaron máquinas de soporte vectorial con funciones *kernel* lineal, polinómica y de base radial. Las clases de Scikit-learn empleadas fueron SVC y LinearSVC, explicadas a continuación.

El objeto SVC emplea un enfoque “uno contra uno” para la clasificación multiclase, de modo que en total se construyen *n_clases*(n_clases-1)/2* clasificadores y cada uno entrena datos de dos clases. Por otro lado, LinearSVC implementa la estrategia “uno contra el resto”, entrenando así los modelos de *n_clases* (Scikit-learn developers, 2020e).

LinearSVC es similar a SVC con parámetro *kernel* = *‘linear’*, pero implementado en términos de liblinear en lugar de libsvm. Por esta razón, tiene mayor flexibilidad en la elección de penalizadores y funciones de pérdida y debería escalar mejor para un número grande de muestras.

En el caso de LinearSVC se ajustó el parámetro de regularización *C*, parámetro que es inversamente proporcional a la fuerza de regularización. Los valores asignados a este hiperparámetro durante el ajuste se muestran en la tabla 3.

Hiperparámetro	**Valores asignados**					
C	0,01	0,1	1	10	100	1000

Tabla 3. Valores asignados al hiperparámetro C para el clasificador SVM lineal.

Por su parte, para la clase SVC se evaluaron las funciones kernel de base radial (*kernel* = *‘rbf’*) y polinómica (*kernel* = *‘polynomial’*).

Al entrenar una SVM con función kernel de base radial (*Radial Basis Function* o RBF) (ecuación 3) deben considerarse dos hiperparámetros: *C* y *gamma*. El hiperparámetro *C*, común para todos los *kernels* de máquinas de soporte vectorial, intercambia la precisión de la clasificación con la simplicidad de la superficie de decisión. Un valor bajo de *C* suaviza la superficie de decisión, mientras un valor alto de *C* tiene como objetivo clasificar todos los ejemplos de

entrenamiento correctamente. Por otro lado, *gamma* define cuánta influencia tiene un solo ejemplo de entrenamiento. Cuanto mayor sea la *gamma*, más cercanos deben estar los otros ejemplos para que se vean afectados (Scikit-learn developers, 2020b).

rbf: $exp\,(-\gamma\|x-x'\|^2)$, donde γ viene dado por el parámetro *gamma* y debe ser mayor que 0 (3)

El comportamiento del modelo es muy sensible al hiperparámetro *gamma*. Si *gamma* es demasiado grande, el radio del área de influencia de los ventores de soporte solo incluye al vector de soporte en sí y ninguna regularización con *C* podrá evitar el sobreajuste. Cuando *gamma* es muy pequeña, el modelo está demasiado restringido y no puede capturar la "forma" de los datos. Los modelos suaves (valores de *gamma* más bajos) pueden hacerse más complejos aumentando la importancia de clasificar cada punto correctamente (valores de *C* más altos) (Scikit-learn developers, 2020b).

Para ajustar este modelo, en primer lugar, se realizó una búsqueda del valor de *gamma*, con un valor de *C* fijo. Después, se utilizó la búsqueda en rejilla con los parámetros *C* y *gamma (tabla 4)*.

Hiperparámetro	**Valores asignados**					
C	0,01	0,1	1	10	100	1000
γ	0,01	0,1	1	10	100	

Tabla 4. Valores asignados los hiperparámetros C y gamma para el clasificador SVM con kernel RBF.

Por último, en el modelo de SVM con función *kernel* polinómica (ecuación 4), los hiperparámetros ajustado fueron el grado del polinomio (*degree*) y la inversa de la fuerza de regularización, *C*.

polynomial: $(\gamma\langle x, x'\rangle + r)^d$, donde d viene dado por el parámetro *degree* y r por *coef0*. (4)

Cuando *d* = 1 y *C* = 0, el resultado es el de una función *kernel* lineal. Si *d* > 1, se generan límites de decisión no lineales, aumentando la no linealidad a medida que aumenta *d*. No es recomendable emplear valores de *d* muy elevados porque puede dar lugar a sobreajuste. En la tabla 5 se muestran los valores asignados a estos parámetros.

Hiperparámetro	**Valores asignados**					
C	0,01	0,1	1	10	100	1000
d	2	3	4	5	6	

Tabla 5. Valores asignados los hiperparámetros C y degree para el clasificador SVM con kernel polinómico.

5.1.2 Random Forest

El algoritmo *random forest* es un método de aprendizaje conjunto, compuesto por múltiples árboles de decisión. Al promediar el impacto de varios árboles de decisión, *random forest* tiende a mejorar la predicción (Huneycutt, 2018).

Los árboles de decisión son un método de aprendizaje supervisado no paramétrico cuya finalidad es crear un modelo que prediga el valor de una variable objetivo mediante el aprendizaje de reglas de decisión simples deducidas de las características de los datos (Scikit-learn developers, 2020e). Un árbol de decisión se dibuja al revés, con su raíz en la parte superior. Los nodos internos son las condiciones en función de las cuales el árbol se divide en ramas (borde). La decisión es el final de la rama que ya no se divide (hoja) (Gupta, 2017)

La función RandomForestClassifier de Scikit-learn ajusta varios clasificadores de árboles de decisión en varias submuestras de conjuntos de datos y usa el promedio para mejorar la precisión predictiva y controlar el sobreajuste (Scikit-learn developers, 2020a).

Para este modelo, se ajustaron los siguientes hiperparámetros (Huneycutt, 2018):

- **n_estimators:** número de árboles
- **max_features:** número máximo de características consideradas para dividir un nodo
- **max_depth:** número máximo de niveles en cada árbol de decisión
- **min_samples_split:** número mínimo de puntos de datos colocados en un nodo antes de dividirlo.
- **min_samples_leaf:** número mínimo de puntos de datos permitidos en una hoja.
- **bootstrap:** método para muestrear los puntos de datos (con o sin reemplazo).

Como se ha indicado anteriormente, se realizó una búsqueda aleatoria inicial con la función RandomizedSearchCV. Los valores asignados a los hiperparámetros ajustados, se exponen en la tabla 6.

Hiperparámetro	Valores asignados											
n_estimators	200	400	600	800	1000	1200	1400	1600	1800	2000		
max_features	'auto'	'sqrt'										
max_depth	10	20	30	40	50	60	70	80	90	100	110	None
min_samples_split	2	5	10									
min_samples_leaf	1	2	4									
bootstrap	True	False										

Tabla 6. Valores asignados a los hiperparámetros del clasificador Random Forest.

Una vez reducido el rango de cada hiperparámetro, se supo donde concentrar la búsqueda y se llevó a cabo una búsqueda de rejilla con la función GridSearchCV.

5.1.3 Regresión logística

La regresión logística es un algoritmo común de clasificación y un método de regresión útil para resolver problemas de clasificación binaria. La regresión logística multinomial es una extensión de la regresión logística que agrega soporte para los problemas de clasificación multiclase (Brownlee, 2021), como es el caso de estudio.

En la regresión logística se utiliza la función sigmoide para asignar un valor entre 0 y 1 a cada punto del espacio de características. La etiqueta asignada a cada punto dependerá del valor que tome la función. Por encima de un valor umbral (generalmente 0.5) se le asigna un valor positivo y, por debajo, un valor negativo (Rodríguez, 2019),

Este algoritmo, por defecto, se limita a problemas de clasificación binaria. Un enfoque para adaptarlo a problemas de clasificación de clases múltiples es dividir el problema en varios problemas de clasificación binaria y ajustar un modelo de regresión logística estándar en cada subproblema. Extensiones como “uno contra uno” (*one-vs-one*) o “uno contra el resto” (*one-vs-rest*) son técnicas de este tipo (Brownlee, 2021).

Un enfoque alternativo implica cambiar el modelo de regresión para predecir la probabilidad de que una entrada pertenezca a cada etiqueta de clase conocida. La distribución de probabilidad que define las probabilidades multiclase se denomina distribución de probabilidad multinomial. Cambiar la regresión logística de binomial a multinomial implica un cambio en la función de pérdida para entrenar el modelo y un cambio en la salida de un valor de probabilidad único a una probabilidad por cada etiqueta de clase (Brownlee, 2021).

El hiperparámetro ajustado fue la inversa de la fuerza de regularización, *C* (tabla 7). Al igual que ocurre con las máquinas de soporte vectorial, a valores más pequeños, regularizaciones más fuertes.

Hiperparámetro	**Valores asignados**					
C	0,0001	0,001	0,01	0,1	1	10

Tabla 7. Valores asignados al hiperparámetro C para la regresión logística multinomial.

5.2 Clasificador óptimo

Una vez se encontraron los parámetros de cada uno de los clasificadores, el siguiente paso fue validar los resultados. De nuevo, se ejecutó una validación cruzada de tipo K-Fold, empleando únicamente los datos de entrenamiento para ajustar los métodos de normalización y extracción de características. El conjunto de validación se procesó empleando los métodos ajustados solo con los datos de entrenamiento antes de utilizarlos en la evaluación de los clasificadores.

Se entrenaron los mejores clasificadores obtenidos con cada uno de los vectores de características y se compararon los resultados para la obtención de un clasificador óptimo. La figura 10 presenta un esquema de la metodología utilizada para encontrar el clasificador óptimo, incluyendo los pasos correspondientes al diseño de los clasificadores mediante el ajuste de hiperparámetros.

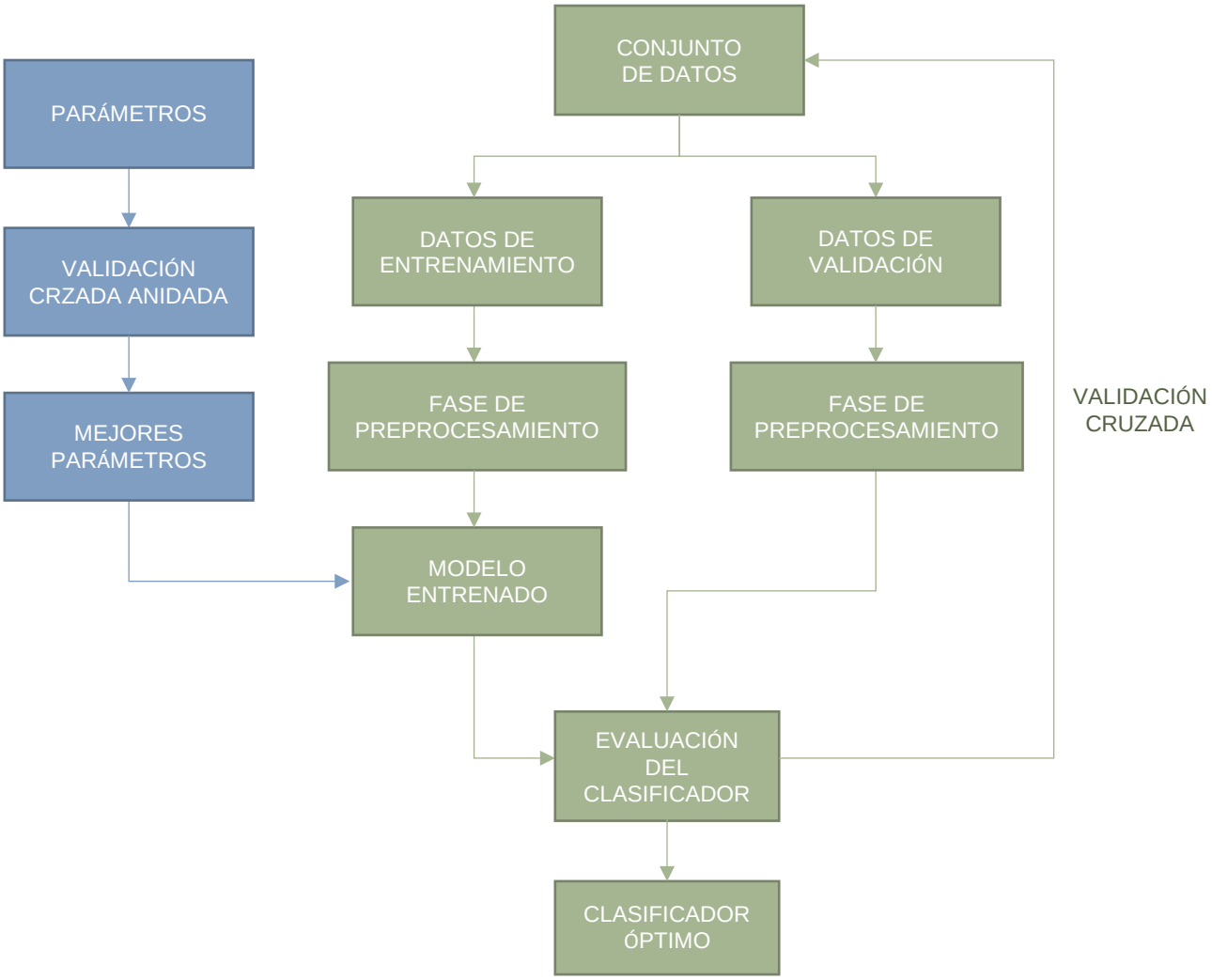


Figura 10. Esquema de la metodología para la obtención del clasificador óptimo. Las etapas en azul se corresponden al proceso descrito en el apartado 3.1, mientras que las etapas en verde se corresponden con el procedimiento expuesto en este apartado.

Es necesario destacar que en este trabajo se replica el procedimiento seguido por Cano et al. 2013, tal y como se ha indicado en epígrafes anteriores. No obstante, estos procedimientos se han simplificado no utilizando un iterador de validación cruzada *LeaveOneOut* (LOO) y aplicando la función *kernel* solo en los clasificadores. Aun así, estas podrían implementarse en el caso de obtener bajos resultados. También podría considerarse aumentar el número de componentes PCA.

En la estrategia de validación cruzada LeaveOneOut (LOO) utilizada en el estudio de Cano et al. (2013), cada conjunto de entrenamiento se crea tomando todas las muestras excepto una, siendo el conjunto de prueba la muestra que se deja fuera. Por lo tanto, para *n* muestras, habrá *n* conjuntos de aprendizaje y *n* conjuntos de validación diferentes. Este procedimiento no desperdicia muchos datos, ya que solo se elimina una muestra del conjunto de entrenamiento (Scikit-learn developers, 2011).

Por otro lado, el método de extracción de características KPCA (*Kernel Principal Component Analysis*), es una extensión del método PCA que logra una reducción de dimensionalidad no lineal mediante el uso de *kernels* (Scikit-learn developers, 2020b).

6 Resultados

6.1 Imágenes de muestra

La figura 11 expone diferentes muestras de aceite de oliva tras el proceso de secado junto con su contenido de impurezas, mostrando la diversidad en el color del aceite y los patrones formados por los sedimentados.

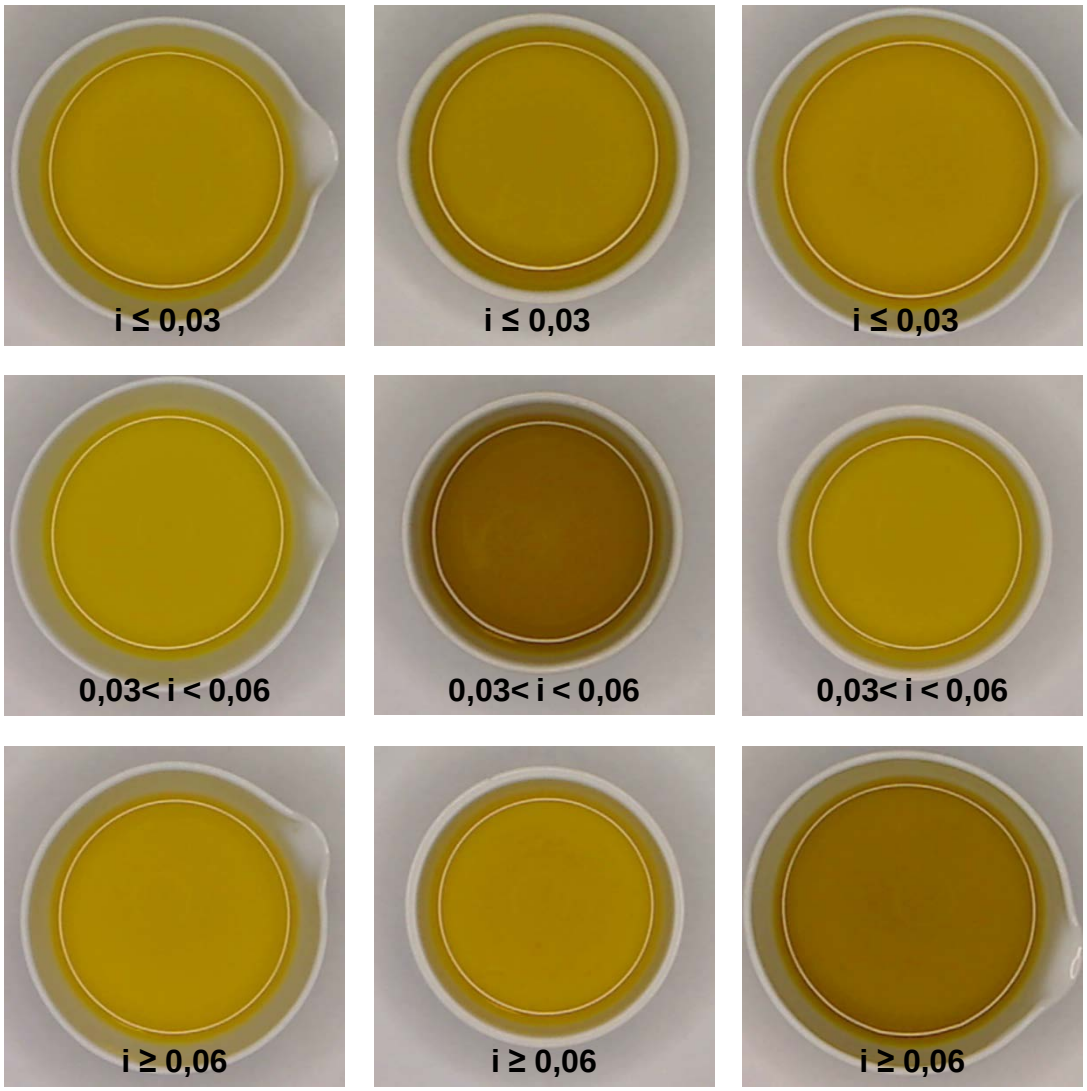


Figura 11. Diferentes muestras de aceite de oliva tras el proceso de secado junto con su contenido de impurezas donde i denota el porcentaje de impurezas en las muestras.

La figura 12 presenta la distribución de los valores del contenido de impurezas de las muestras estudiadas. Estos valores varían ente el 0,02% y el 0,10%. Como se observa en la figura, la mayoría de las muestras tiene un contenido de impurezas no solubles del 0,03%, lo que se corresponde con un nivel bajo.

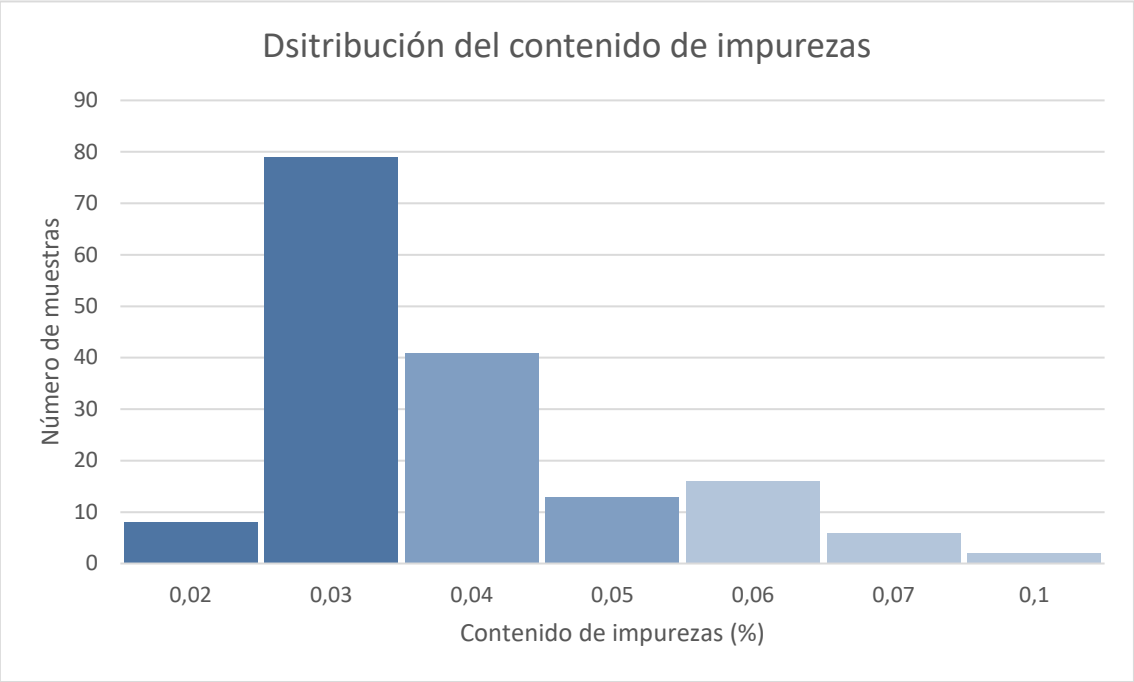


Figura 12. Distribución del contenido de impurezas no solubles para las muestras estudiadas.

6.2 Vector de características

6.2.1 Resultados de la segmentación de imágenes

A continuación, se exponen varios ejemplos del resultado de la segmentación de imágenes (figuras 13 a 17). Se puede comprobar que los rangos establecidos abarcan la variedad de colores que presentan las muestras de aceite. Por otro lado, para eliminar los bordes del recipiente del aceite, se aumentó el valor del límite inferior de la saturación (S). Esto dio lugar a que también desapareciera el aro blanco que aparece en el aceite debido al anillo de iluminación.

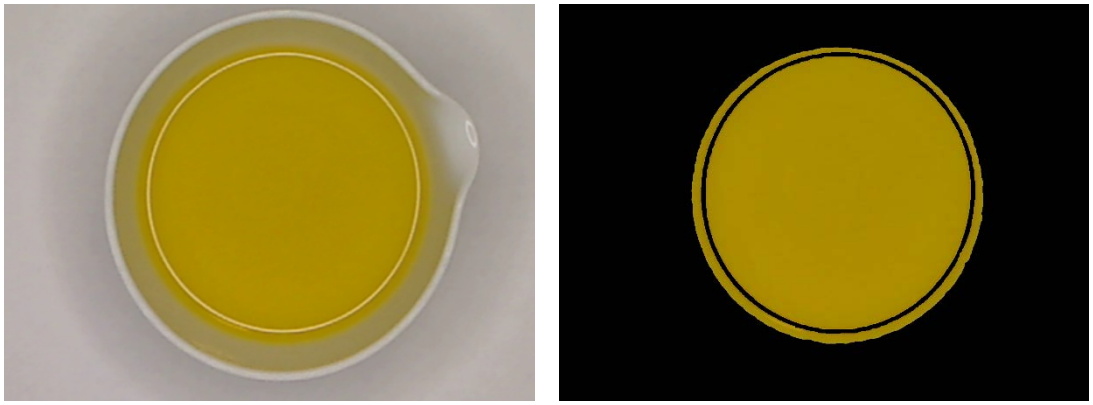


Figura 13. Imagen de muestra i0_04__21 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.04%

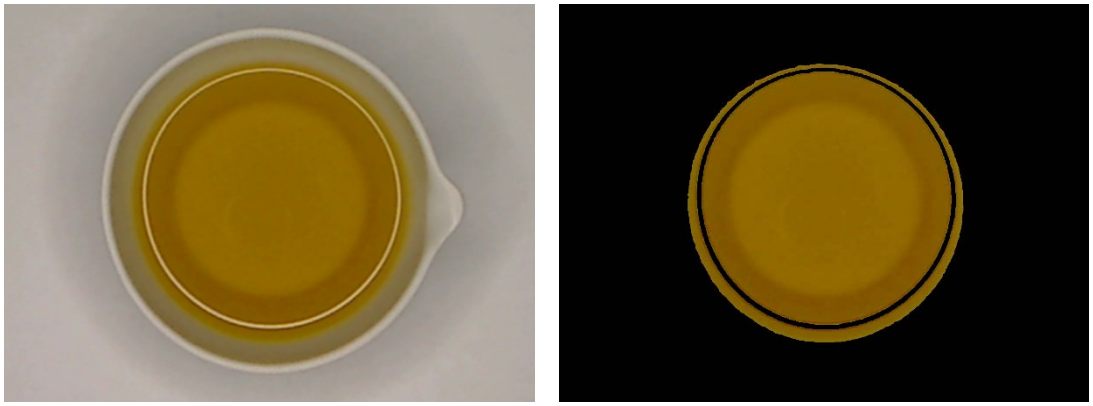


Figura 14. Imagen de muestra i0_04__21 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.04%

Figura 15. Imagen de muestra i0_05__2 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.05%

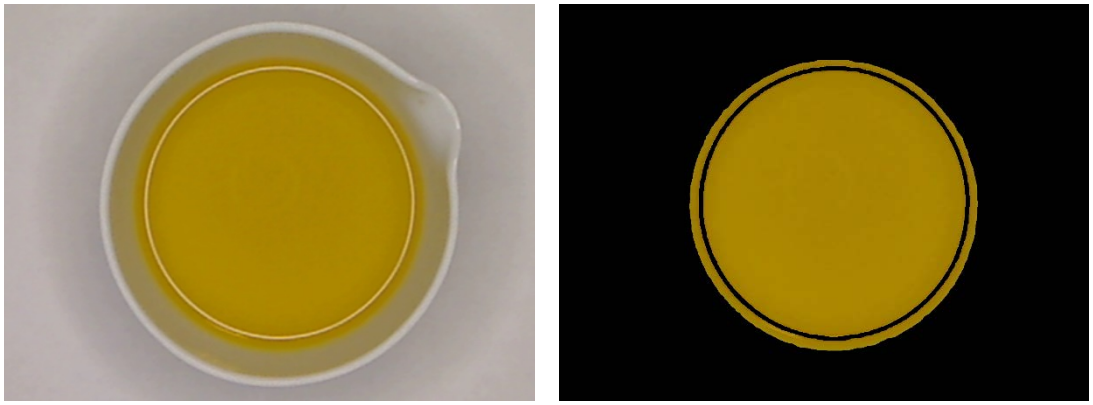


Figura 16. Imagen de muestra i0_06__12 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.06%

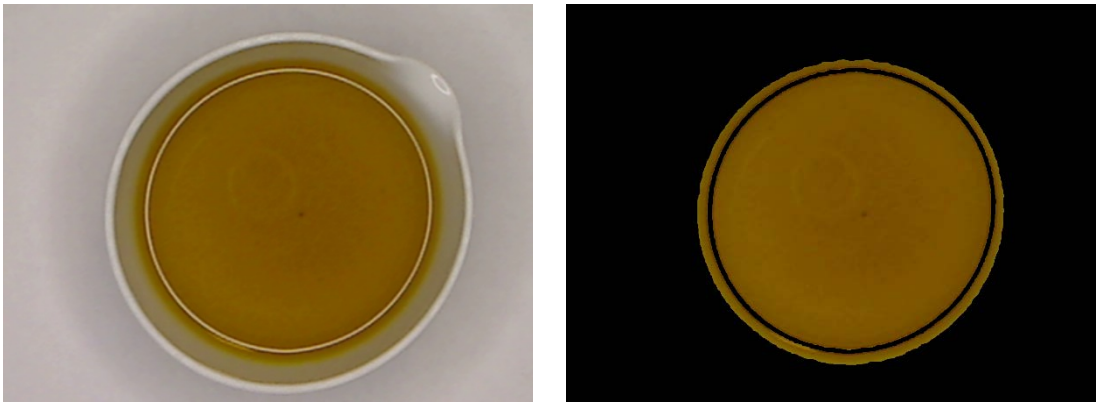


Figura 17. Imagen de muestra i0_10__0 segmentada (derecha) y sin segmentar (izquierda). Esta muestra tiene un contenido de impurezas del 0.10%

6.2.2 Composición del vector de características. Visualización de histogramas.

Empleando las funciones desarrolladas, se representaron múltiples histogramas variando el parámetro de agrupación. En algunos casos, se usó el mismo factor de agrupación en todos los canales; mientras que en otros se establecieron factores de agrupación diferentes para que todos los histogramas tuviesen un número similar de elementos. La tabla 8 muestra el valor de los parámetros empleados en las tres funciones desarrolladas anteriormente. Los histogramas que mostraron mejores resultados fueron los histogramas con reducción de elementos a la cuarta parte (n = 4) obtenidos a partir de las imágenes segmentadas (figuras 18 a 24).

PRUEBA	n	n_H	n_Lab
1	1	1	1
2	2	2	2
3	4	4	4
4	4	3	4
5	8	6	7

Tabla 8. Parámetros de agrupación establecidos para cada canal en las distintas pruebas realizadas.

Como se indicó en el apartado 4.2, inicialmente se consideró como vector de características el ‘set 3’ desarrollado por Cano Marchal, et al. (2013). Los histogramas de los canales que forman este vector se muestran en las figuras 18 a 24. Por otro lado, los histogramas que presentaron una mayor diferenciación entre niveles de impureza fueron los canales G y R del espacio de color BGR, el canal V del espacio de color HSV y los tres canales del espacio de color CIELAB, en menor medida en el caso de los canales a y b.

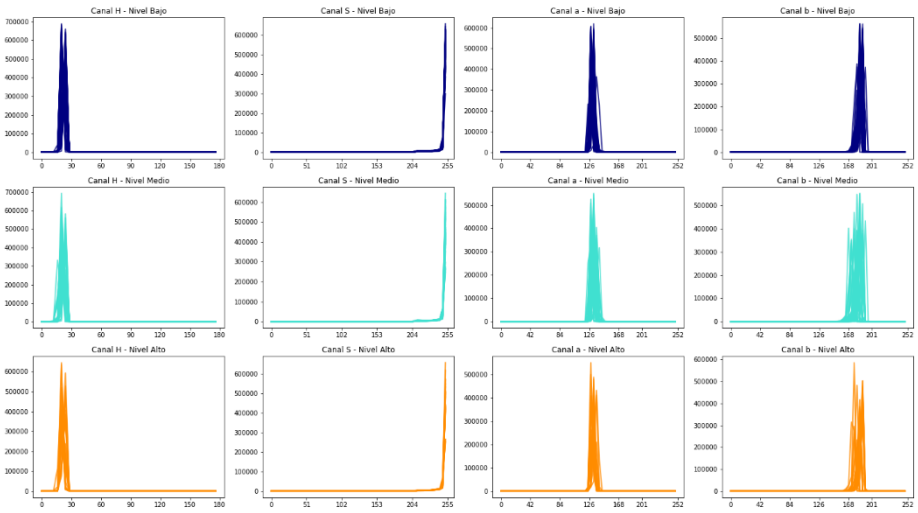


Figura 18. Histogramas agrupados de los canales H, S, a y b de las imágenes de muestra segmentadas.

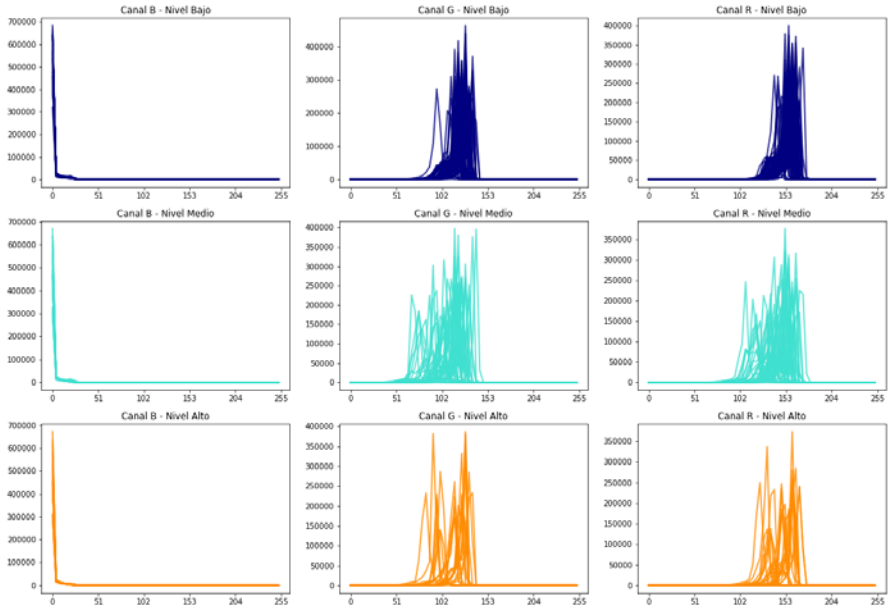


Figura 19. Histogramas agrupados de los canales del espacio de color RGB de las imágenes de muestra segmentadas.

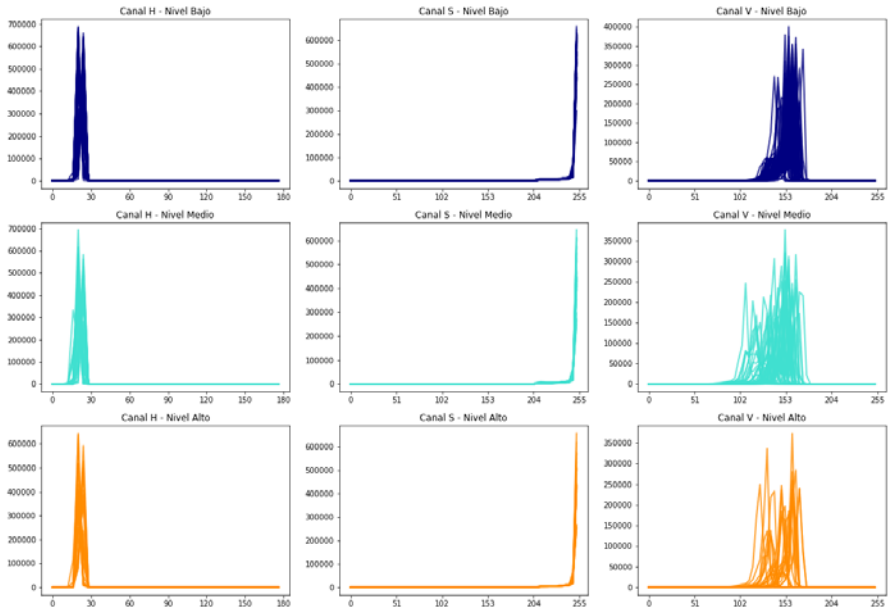


Figura 20. Histogramas agrupados de los canales del espacio de color HSV de las imágenes de muestra segmentadas.

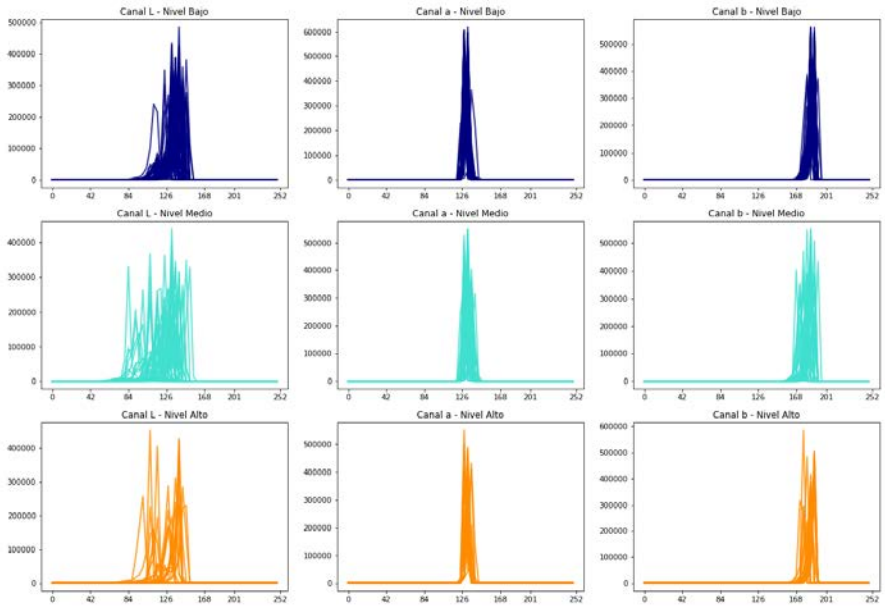


Figura 21. Histogramas sin agrupar de los canales del espacio de color CIELAB de las imágenes de muestra segmentadas.

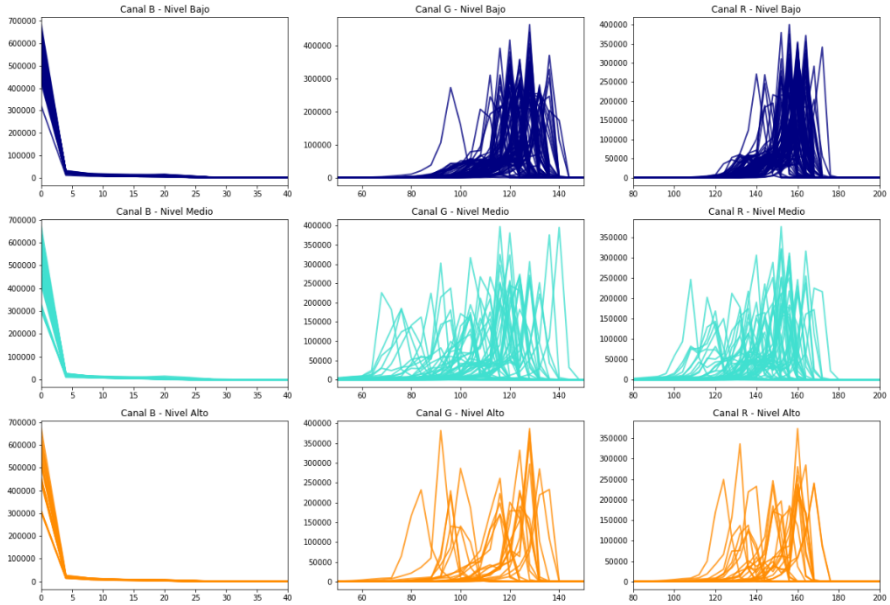


Figura 22. Ampliación de los histogramas agrupados de los canales del espacio de color RGB de las imágenes de muestra segmentadas.

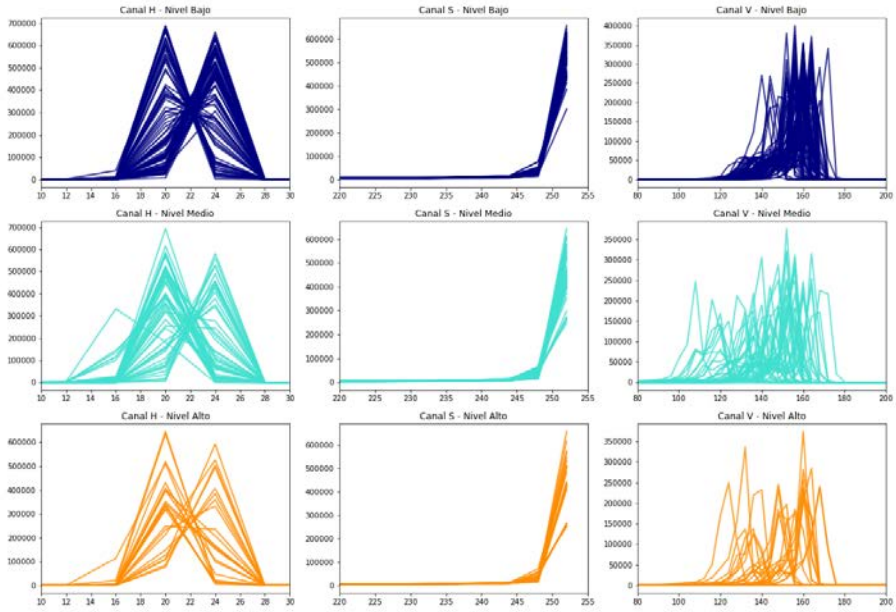


Figura 23. Ampliación de los histogramas agrupados de los canales del espacio de color HSV de las imágenes de muestra segmentadas.

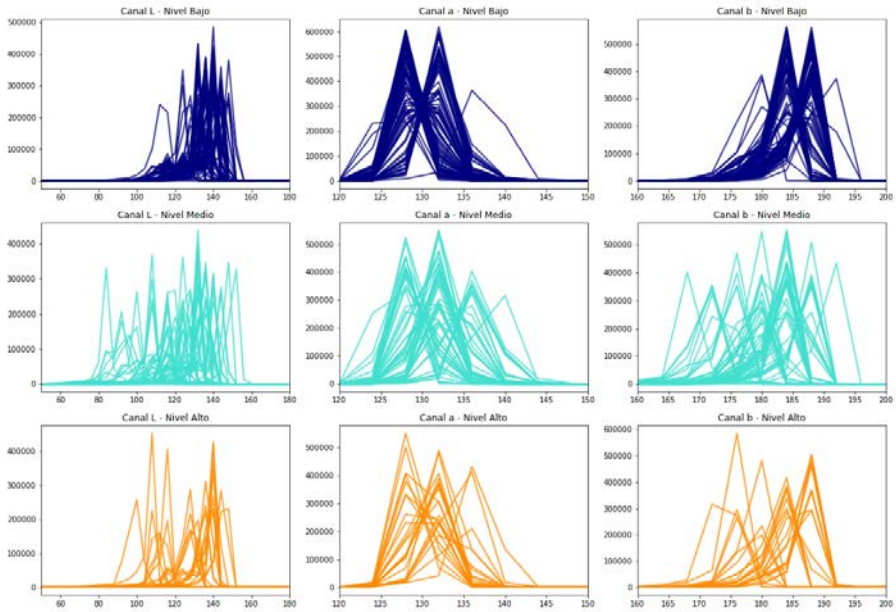


Figura 24. Ampliación de los histogramas agrupados de los canales del espacio de color CIELAB de las imágenes de muestra segmentadas.

6.2.3 Vectores de características configurados

El primer vector de características elaborado, mostrado en la figura 25, estaba formado por los canales *H* y *S* del espacio de color *HSV*, y los canales *a* y *b* de *CIELAB* de acuerdo con el algoritmo definido en la figura 5.

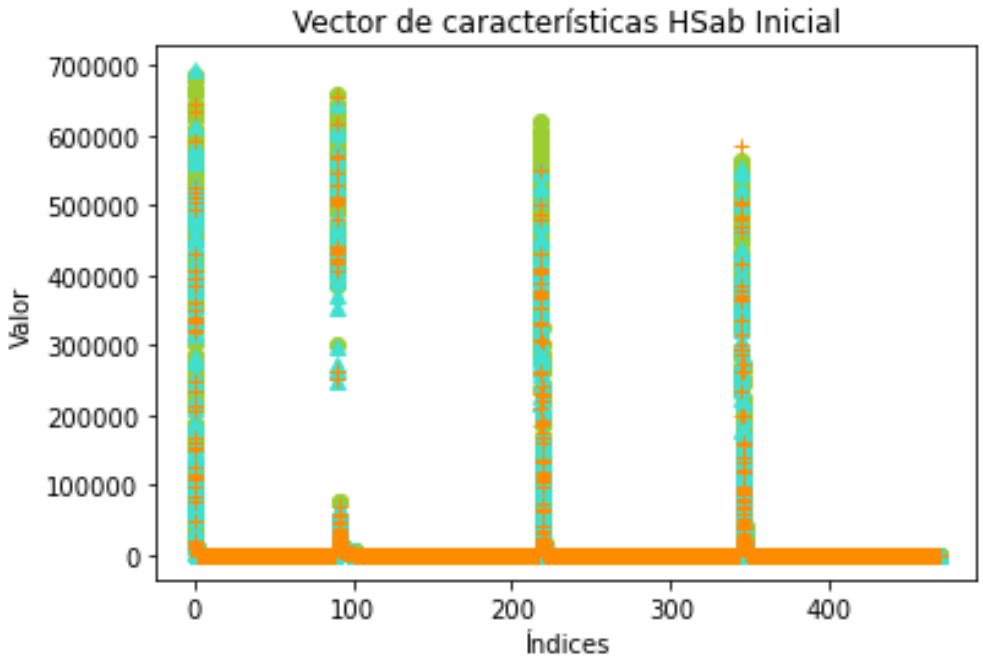


Figura 25. Vector de características HSab Inicial, formado por los canales H, S, a y b transformados según el algoritmo definido en la figura 5

En segundo lugar, se desarrolló un vector de características equivalente al 'set 2' del trabajo de Cano Marchal, et al. (2013). Este vector (figura 26) encadena directamente los histogramas de los canales *H* y *S* de *HSV*, y los canales *a* y *b* de *CIELAB*, sin modificarlos como en el caso anterior.

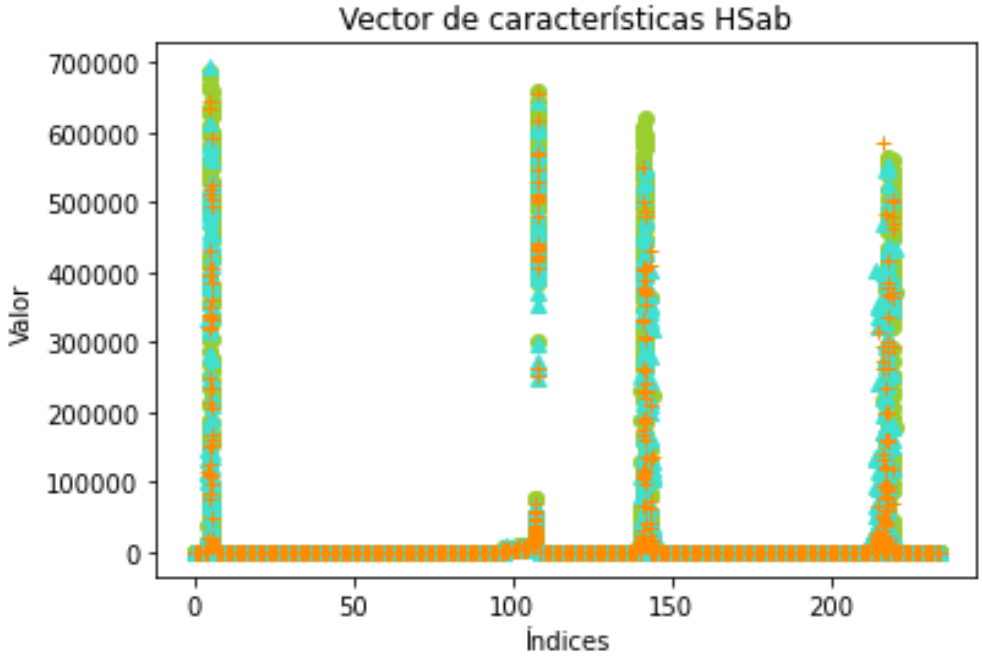


Figura 26. Vector de características HSab, formado por los canales H, S, a y b.

Después del análisis visual de los histogramas representados en el apartado anterior, se construyó un nuevo vector de características (figura 27) incluyendo los canales del espacio de color *CIELAB*, y los canales *G* y *R* del *RGB*.

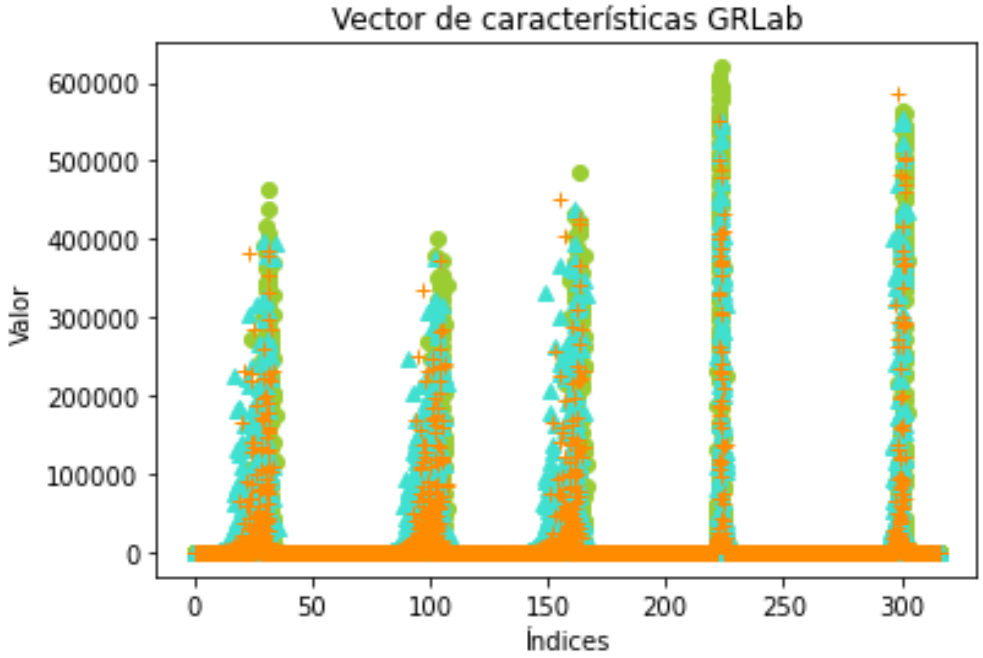


Figura 27. Vector de características GRLab, formado por los canales G, R, L, a y b.

Por último, se elaboró una versión del vector de características anterior que incluyese solo los canales G y R del espacio de color RGB, y el canal L de CIELAB.

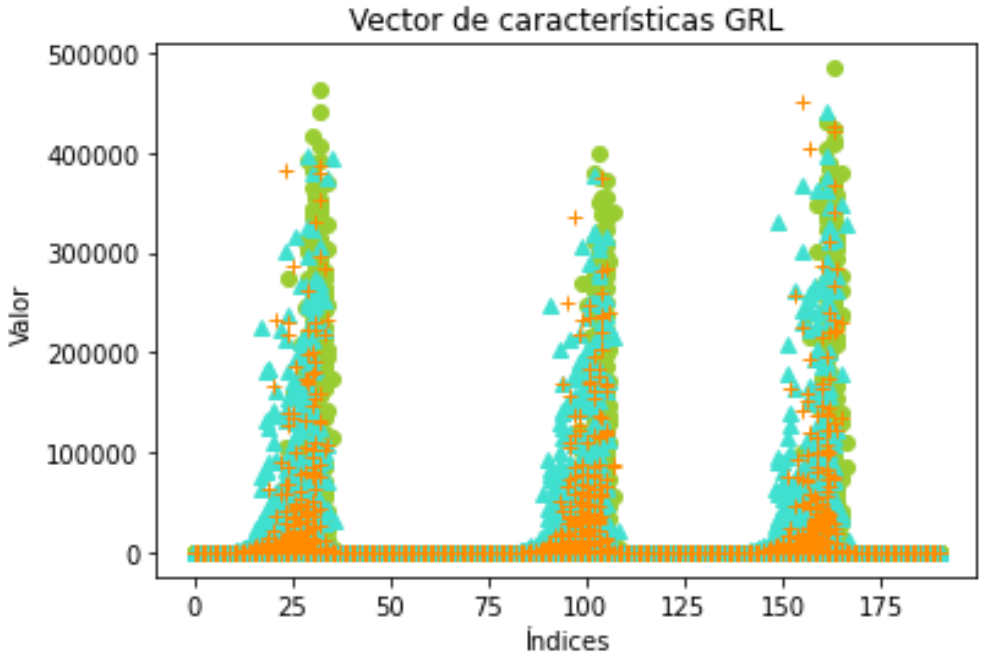


Figura 28. Vector de características GRL, formado por los canales G, R y L.

Como se puede observar en los histogramas representados en el apartado anterior, los canales R y V de los espacios de color RGB y HSV, son muy similares. Por esta razón, solo se tuvo en cuenta el primero a la hora de diseñar las configuraciones de vectores de características.

En la tabla 9 se presenta un resumen de los vectores de características evaluados. Para facilitar la compresión de los resultados, a cada uno se le ha asignado un color, con el que se identificarán las tablas de resultados correspondientes en los siguientes apartados.

Nombre	Descripción	Número de características
HSab inicial	Histogramas de los canales H, S, a y b transformados según el algoritmo descrito en la figura 5	230
HSab	Histogramas de los canales H, S, a y b	235
GRLab	Histogramas de los canales G, R, L, a y b	317
GRL	Histogramas de los canales G, R y L	191

Tabla 9. Vectores de características evaluados.

6.2.4 Resultado de la normalización del vector de características

Los vectores de características se transformaron al intervalo [0,1] y se centraron antes de emplearlos como entrada del método de extracción de características. El resultado de la normalización se expone en la figura 29.

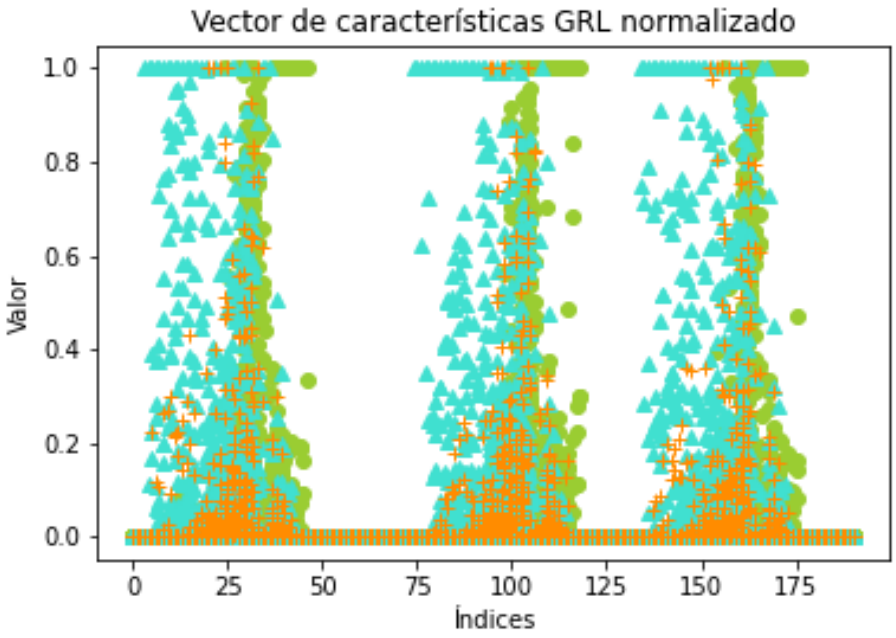


Figura 29. Vector de características GRL normalizado.

6.2.5 Extracción de características. Visualización de la separación entre clases

Tras realizar el análisis de componentes principales (*Principal Component Analysis* o PCA) se visualizó la distribución de las clases en el espacio de características para obtener una comprensión intuitiva de la dificultad de la tarea.

La observación de las distribuciones de las muestras para los distintos vectores de características analizados señaló que los datos no están claramente agrupados de acuerdo con el nivel de impurezas, los que dificulta al clasificador ofrecer resultados satisfactorios con buenas propiedades de generalización.

Se observaron resultados muy similares en las tres configuraciones del vector de características, sin una separación clara entre clases. Los vectores de características con configuración HSab (figura 31) y GRLab (figura 32) dieron los mejores resultados.

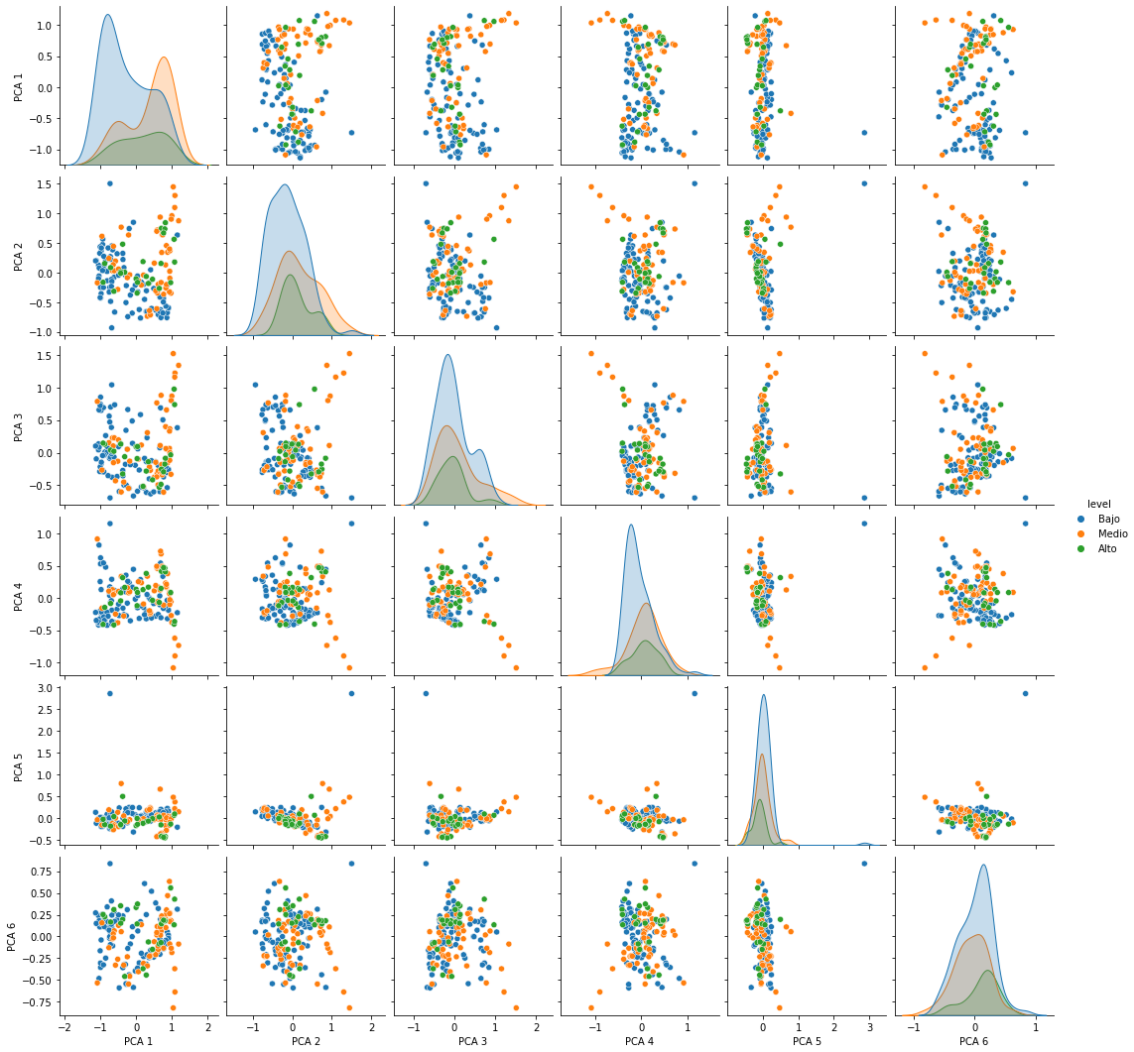


Figura 30. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características HSab.

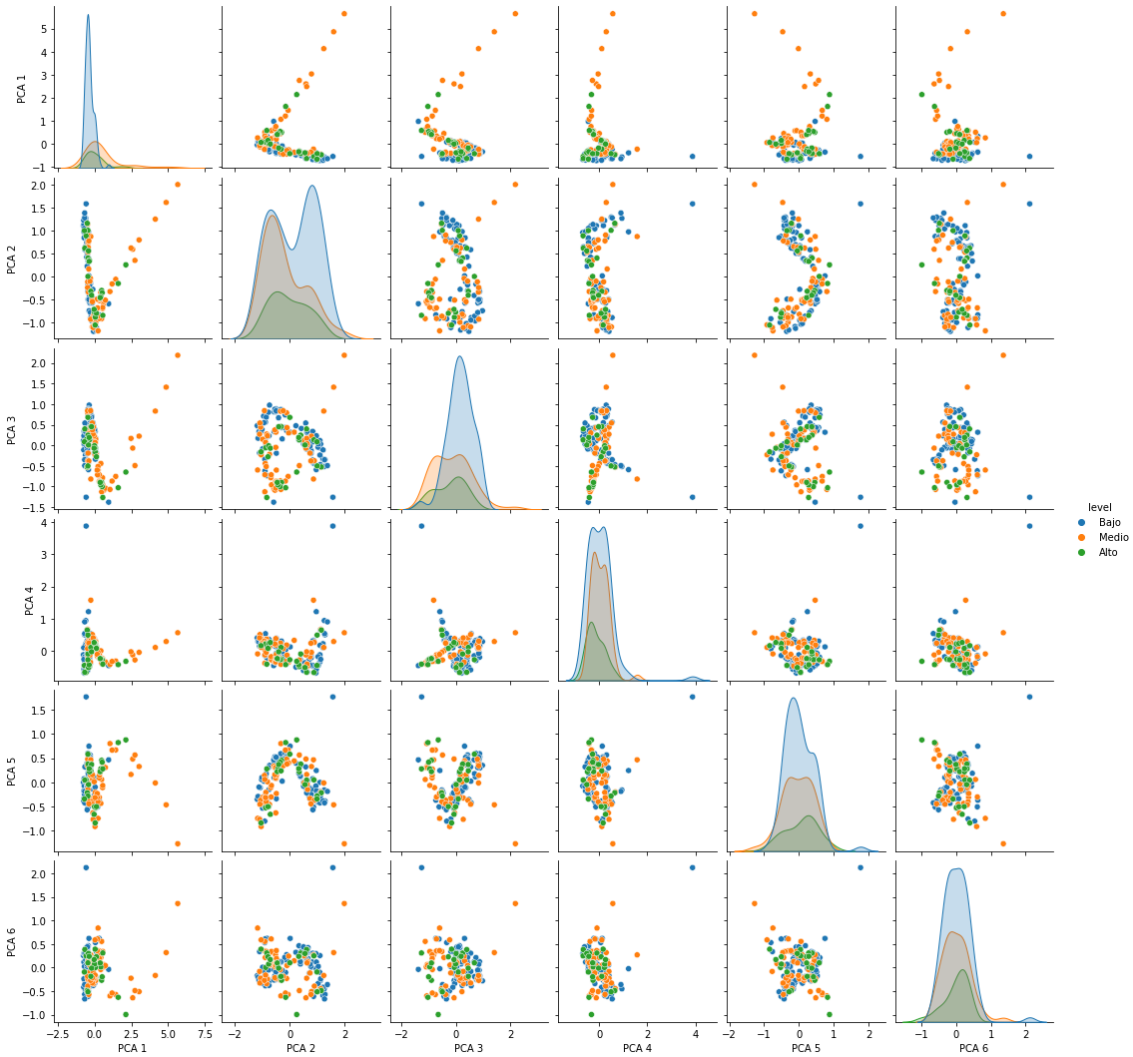


Figura 31. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características GRLab.

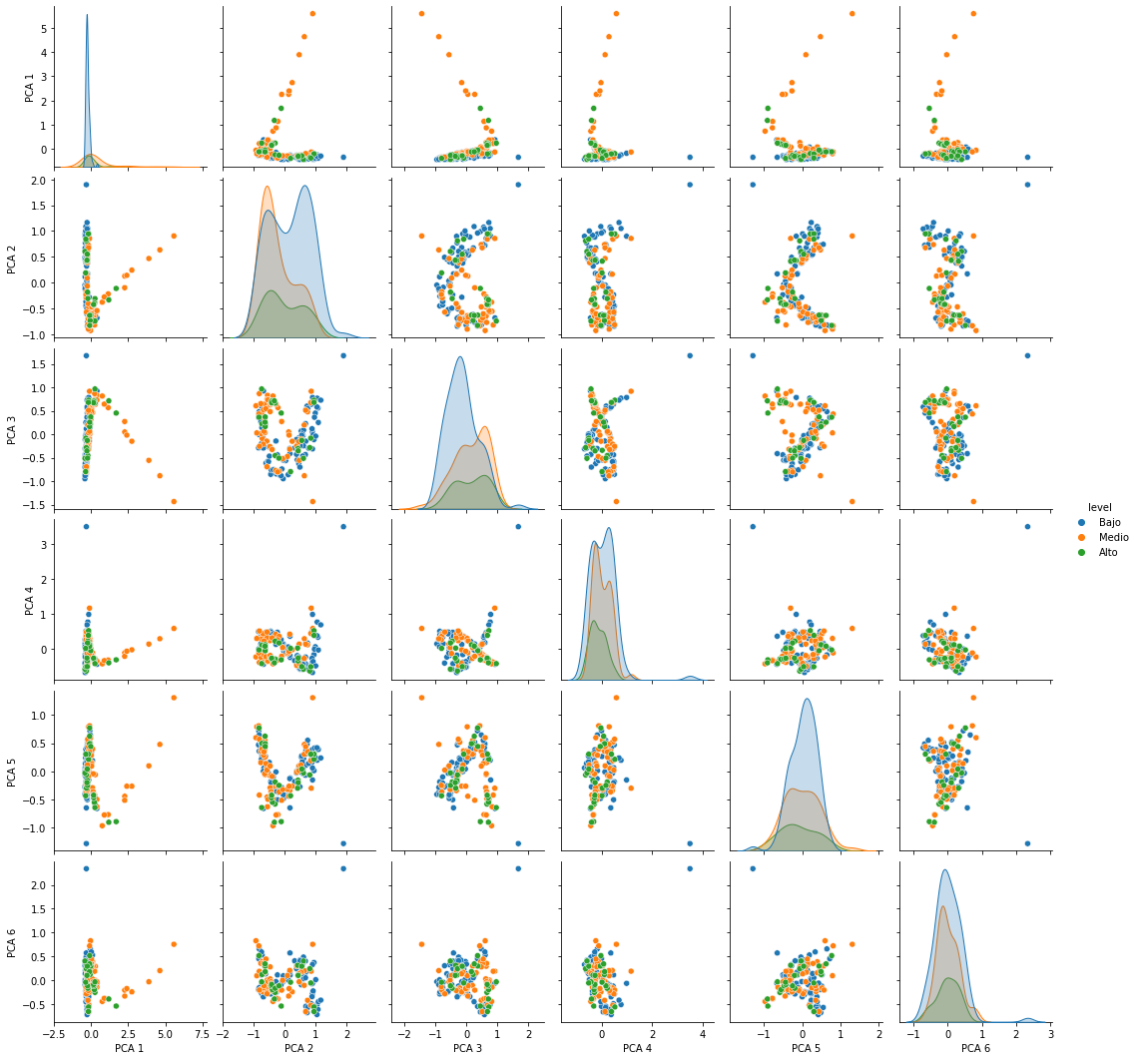


Figura 32. Visualización de las seis componentes de PCA calculadas para todo el conjunto de características. Vector de características GRL.

6.3 Clasificador

6.3.1 Coeficientes de la transformación PCA

Para comprobar la robustez del método de extracción de características, se visualizaron los coeficientes de la transformación PCA de cada una de las características para cada iteración del procedimiento de validación cruzada (figuras 33 a 40).

Como se observa en las gráficas, las componentes proporcionadas por cada subconjunto fueron muy similares en los cuatro vectores de características estudiados. Sin embargo, para algunas componentes hubo subconjuntos que generaron valores semejantes, pero de signo contrario al del resto de subconjuntos. En este sentido, el vector de características HSab Inicial (figuras 33 y 34) es el que dio lugar a resultados menos uniformes, dado que solo la componente discriminativa 1 (primera fila de la figura 33) es similar para todos los subconjuntos. Los vectores de características GRLab (figuras 37 y 38) y GRL (figuras 39 y 40) mostraron mejores resultados. En cuanto a la magnitud de los valores asignados, el vector de características HSab (figuras 35 y 36) es el que otorga los valores pico más elevados en valor absoluto, en comparación al resto de vectores estudiados.

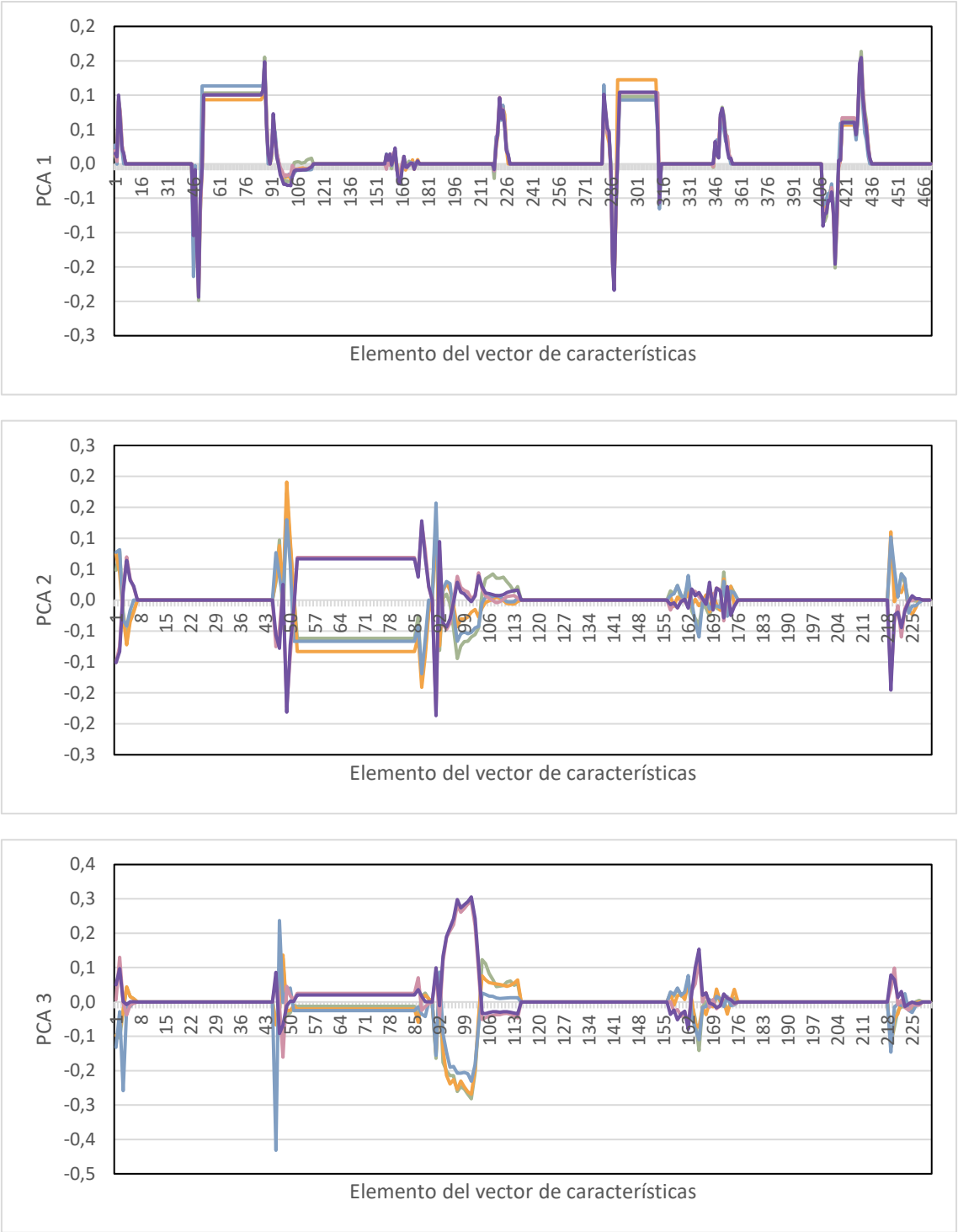


Figura 33. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características HSab Inicial.

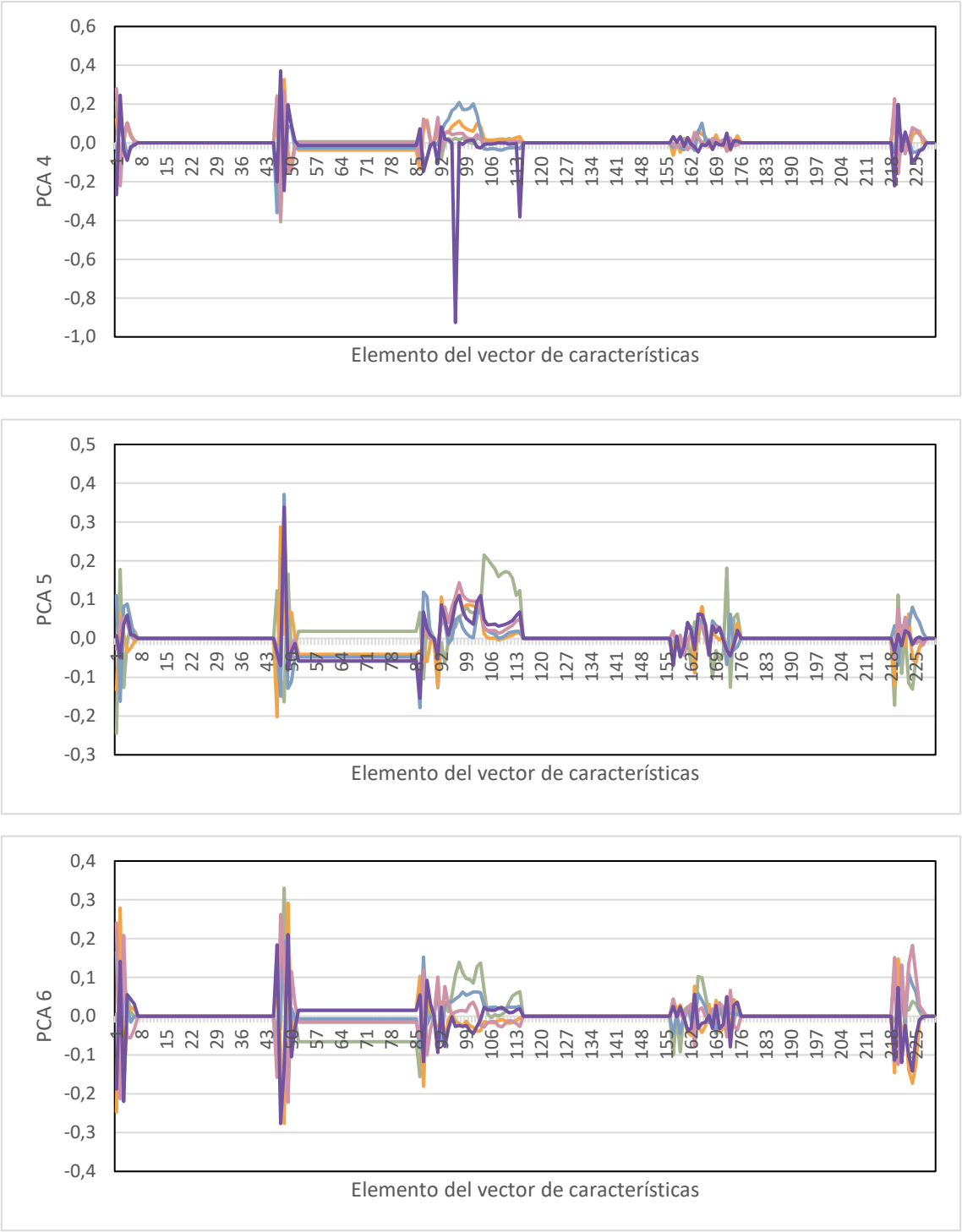


Figura 34. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características HSab Inicial.

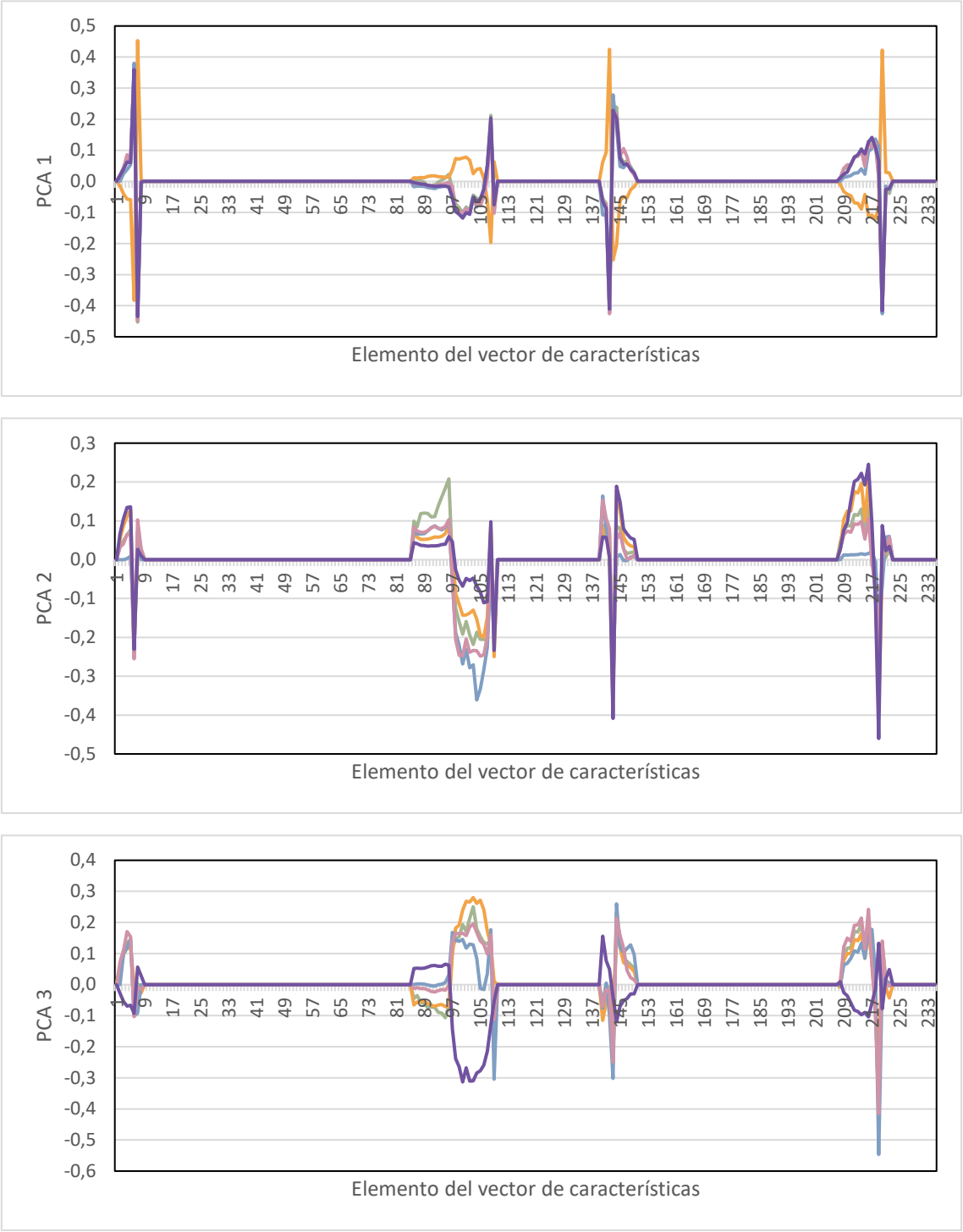
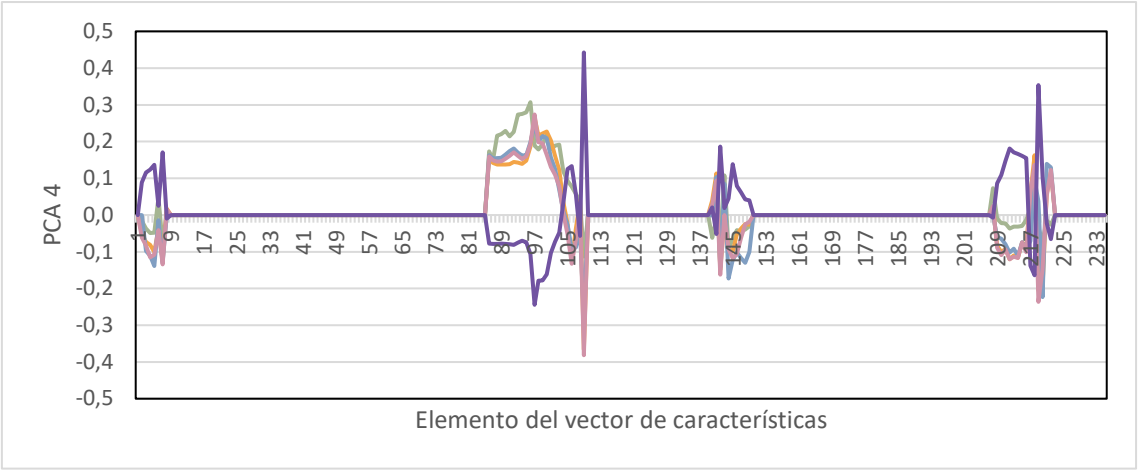
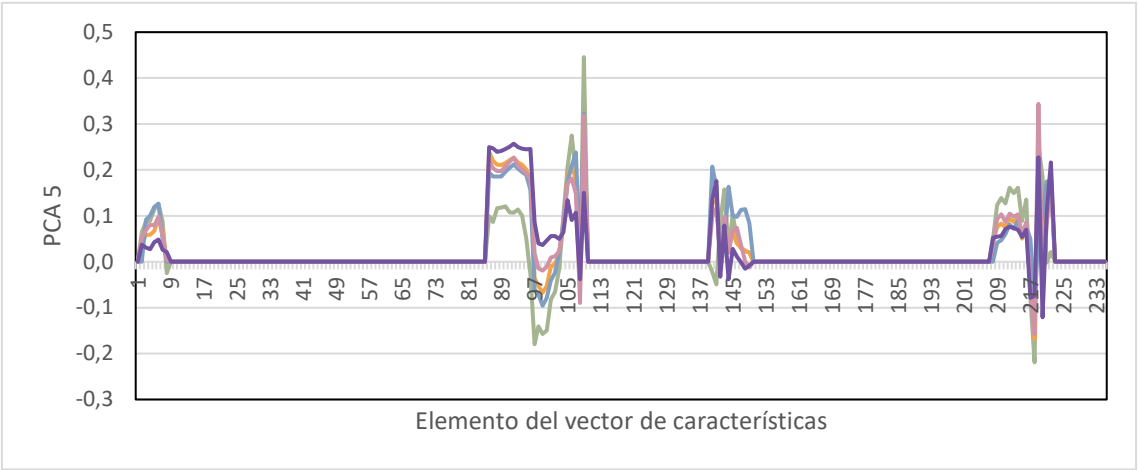


Figura 35. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características HSab.





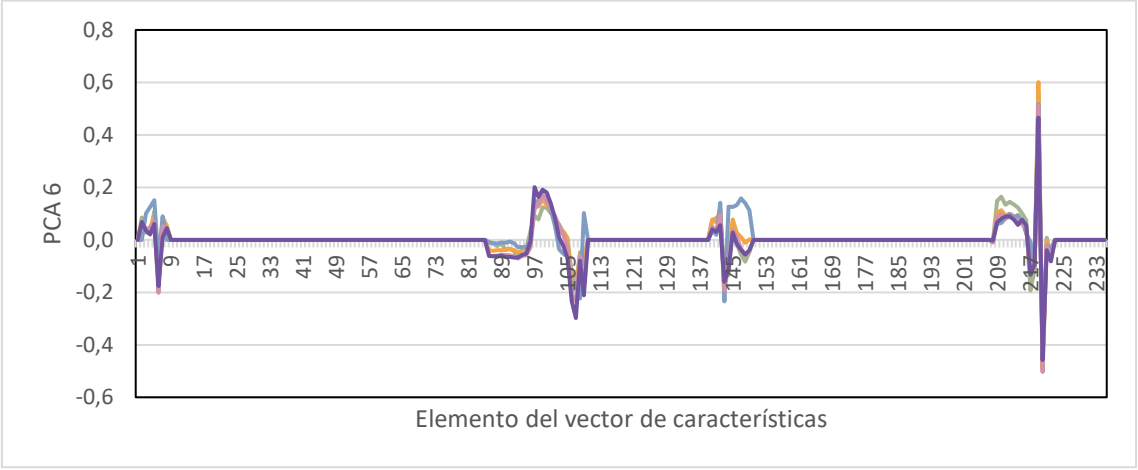


Figura 36. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características HSab.

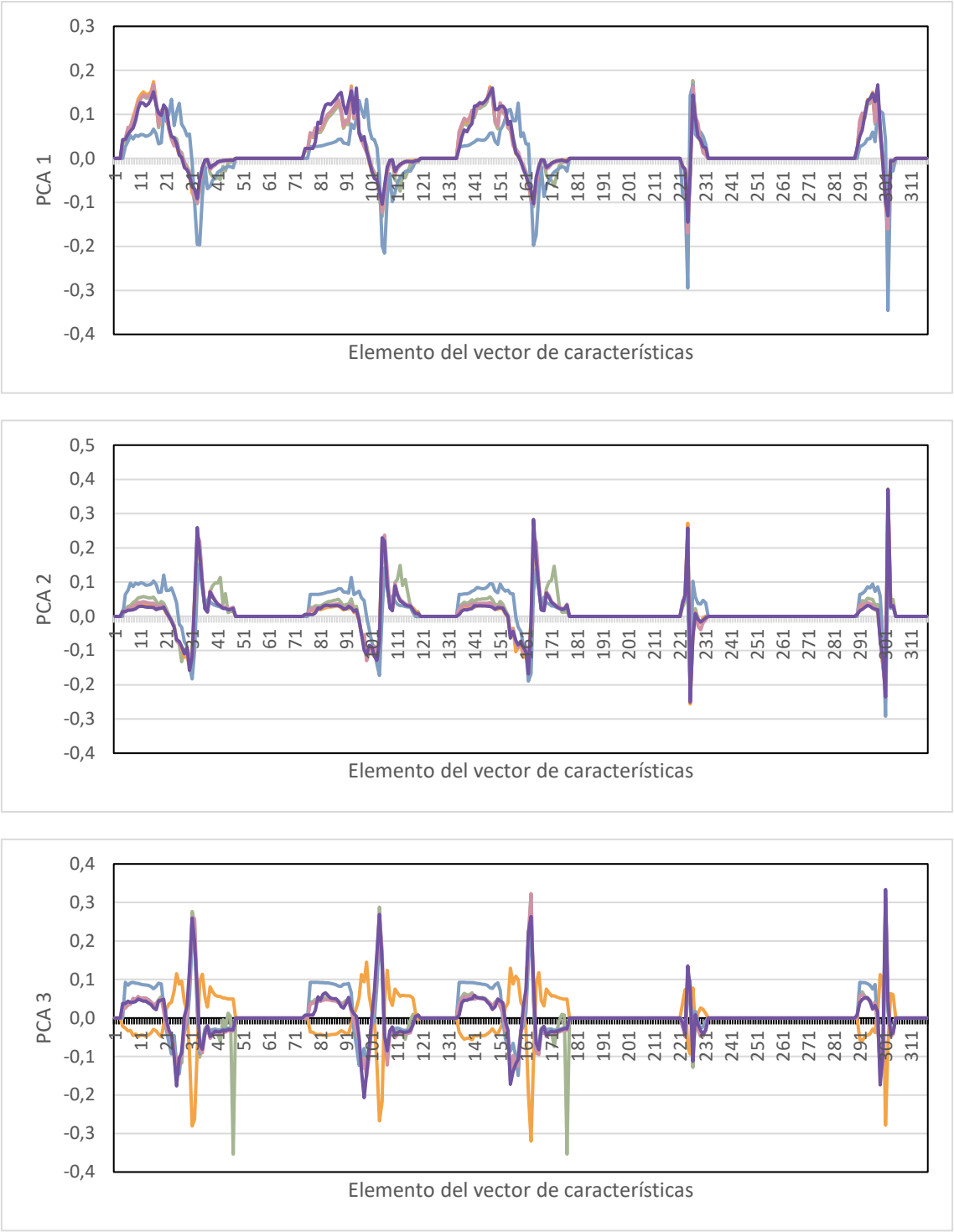


Figura 37. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características GRLab.

Figura 38. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características GRLab.

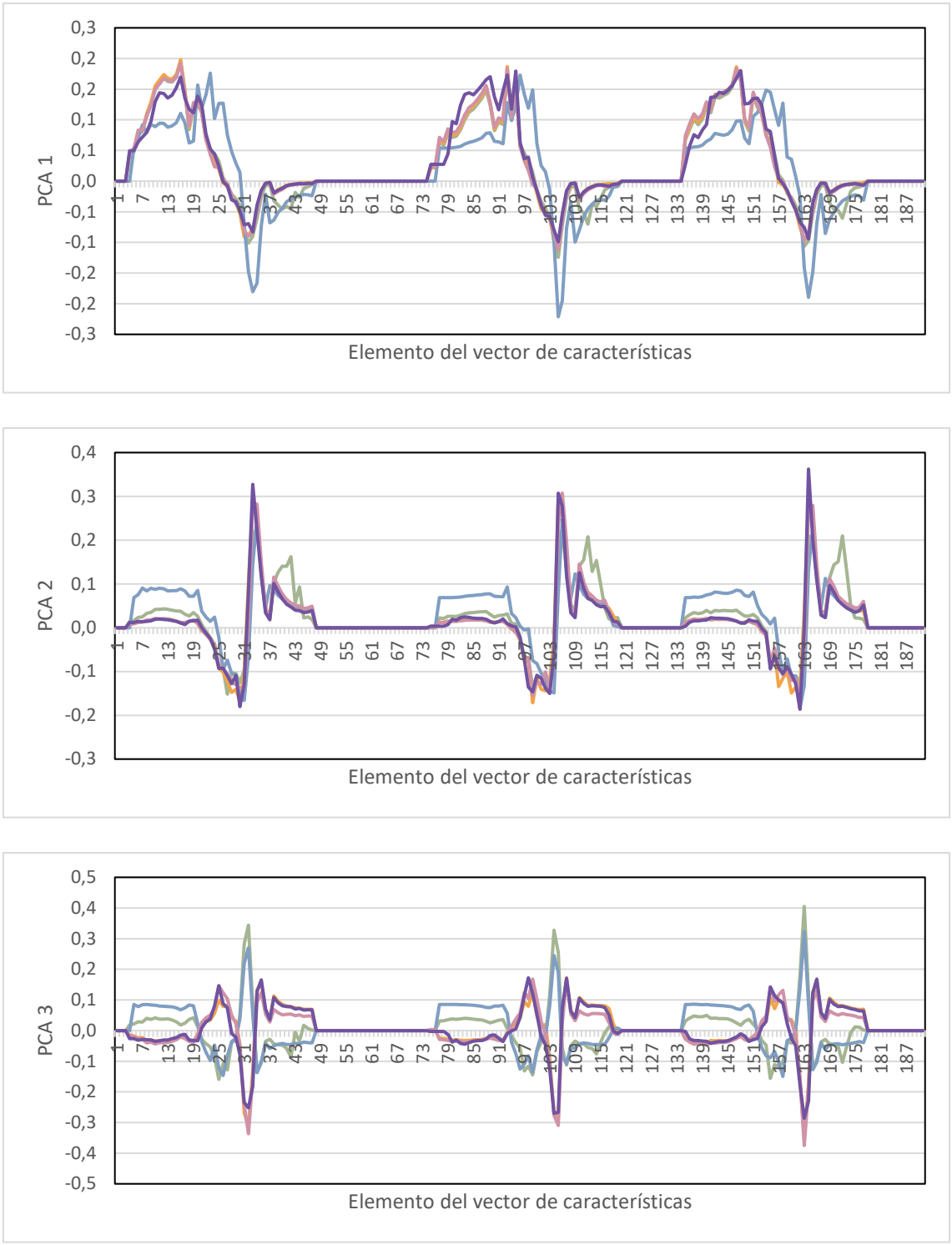
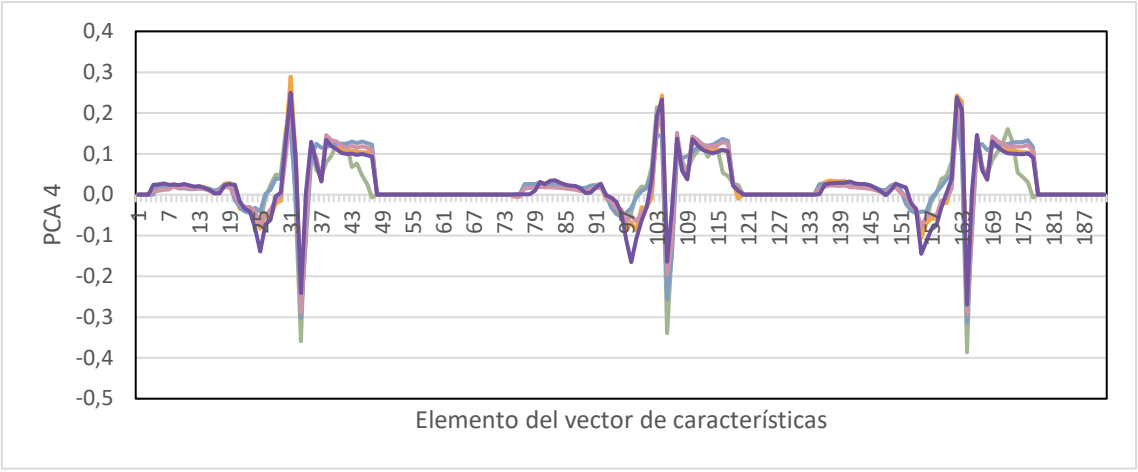
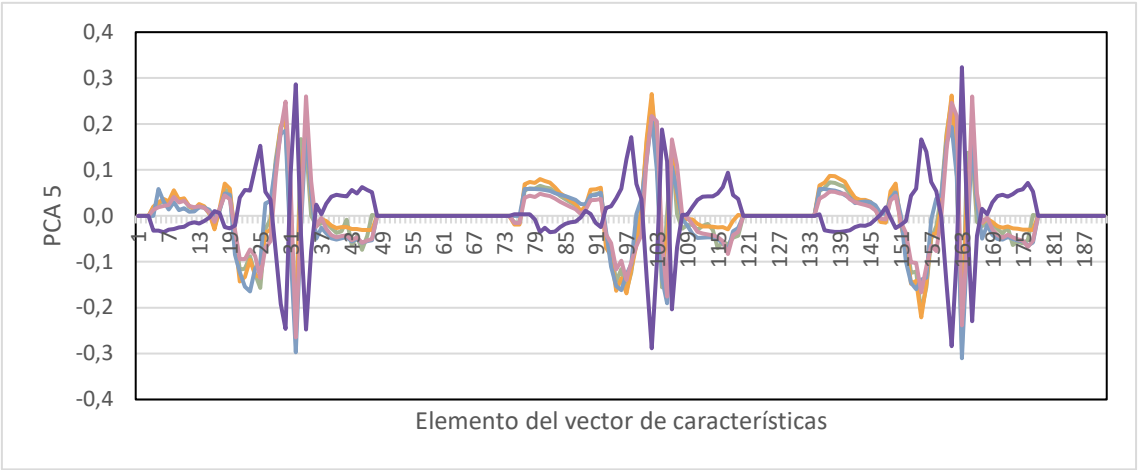


Figura 39. Visualización de los coeficientes de las tres primeras componentes de PCA para el vector de características GRL.





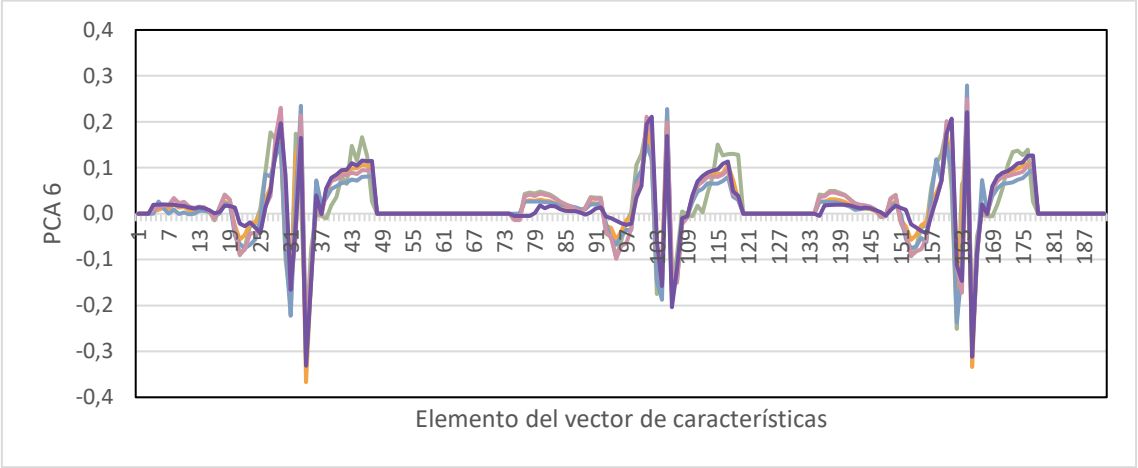


Figura 40. Visualización de los coeficientes de las tres últimas componentes de PCA para el vector de características GRL.

6.3.2 Hiperparámetros obtenidos

A continuación, se exponen los resultados correspondientes al ajuste de los hiperparámetros de cada uno de los clasificadores evaluados.

6.3.2.1 Máquina de Soporte Vectorial Lineal

Para todos los vectores de características estudiados, el valor ajustado del hiperparámetro C fue el mismo en la mayoría de las iteraciones, mostrando un buen nivel de generalización en el ajuste.

Al visualizar la precisión de los modelos de entrenamiento y validación para cada valor de C en cada iteración (figuras 42 a 45) se observó que ambos modelos (entrenamiento y validación) tienen precisiones similares, lo que corroboró que no se estaba produciendo sobreajuste. Sin embargo, el nivel de predicción no es del todo satisfactorio, puesto que es inferior al 70% tanto en el entrenamiento como en la validación. Aunque la precisión de entrenamiento media obtenida fue similar para todos los vectores de características, la mejor precisión de validación se dio para los vectores GRLab (tabla 12) y GRL (tabla 13). El vector de características HSab (tabla 11) tuvo una buena precisión de validación; sin embargo, la elevada desviación típica mostraba una baja capacidad de generalización del modelo, por lo que se descartó este vector como opción de clasificador óptimo. El vector de características HSab Inicial (tabla 10) se descartó también por ser el modelo con menor precisión en la validación.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	1	62,02	48,48
2	1	63,48	39,39
3	0,01	60,4	60,61
4	0,01	61,97	51,52
5	1	62,82	45,45
Media		62,14	49,09
Desviación típica		1,03	7,02

Tabla 10. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características HSab Inicial.

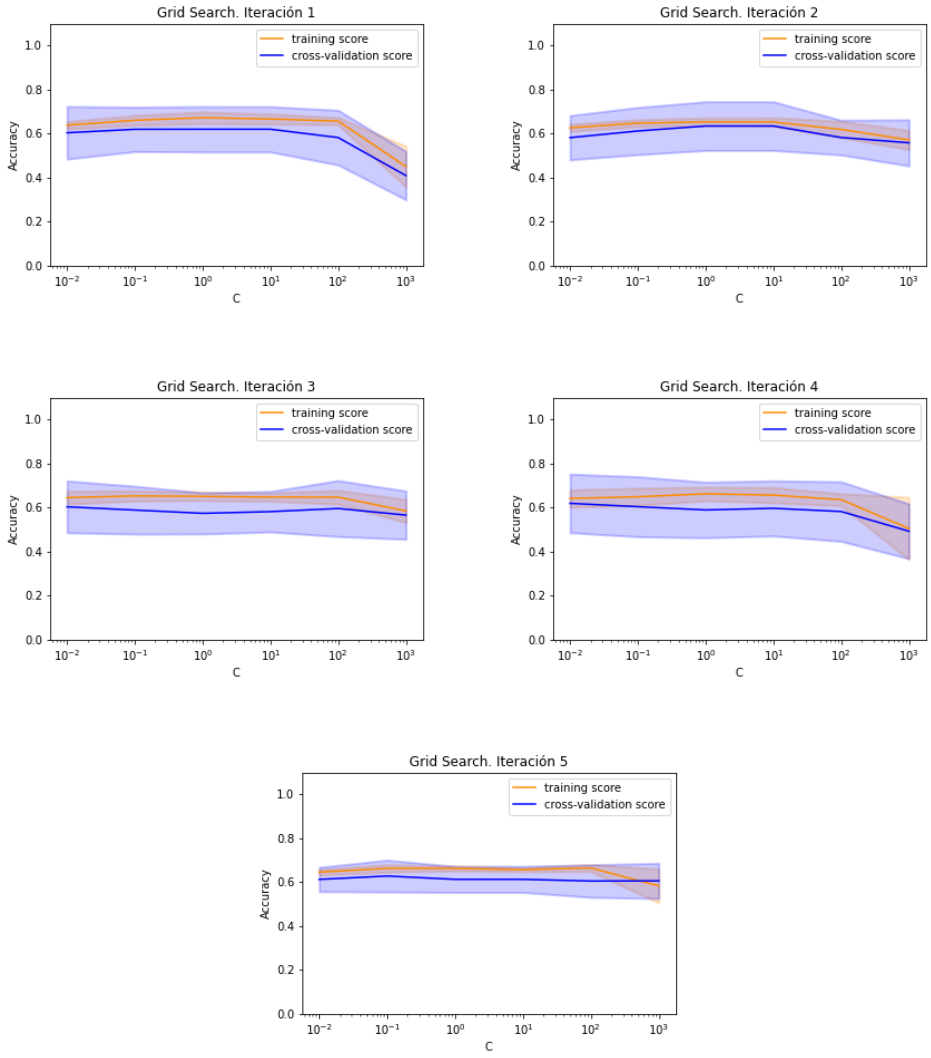


Figura 41. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características HSab Inicial.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	10	65,84	66,67
2	0,1	62,82	39,39
3	0,1	65,81	60,61
4	0,1	65,81	45,45
5	10	66,64	63,64
Media		65,38	55,15
Desviación típica		1,32	10,74

Tabla 11. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características HSab

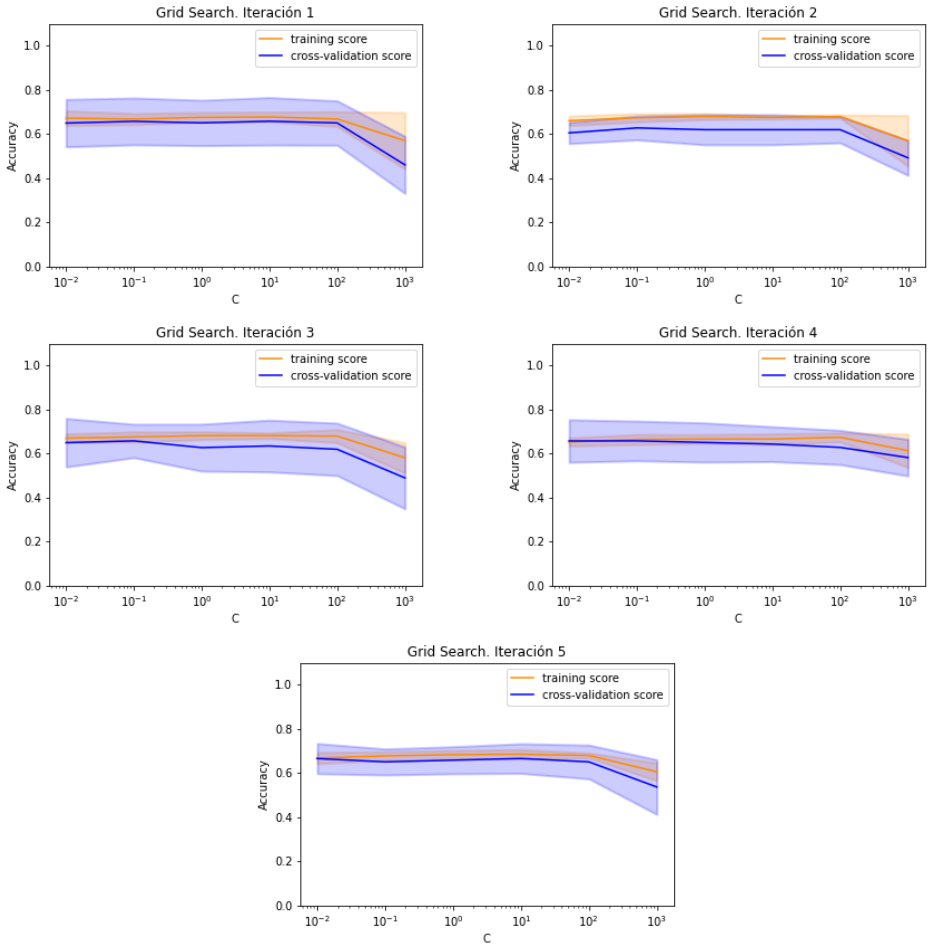


Figura 42. . Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características HSab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	61,99	63,64
2	0,1	62,79	63,64
3	0,01	64,3	63,64
4	0,01	66,55	66,67
5	100	65,78	66,67
Media		64,28	64,85
Desviación típica		1,72	1,48

Tabla 12. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características GRLab.

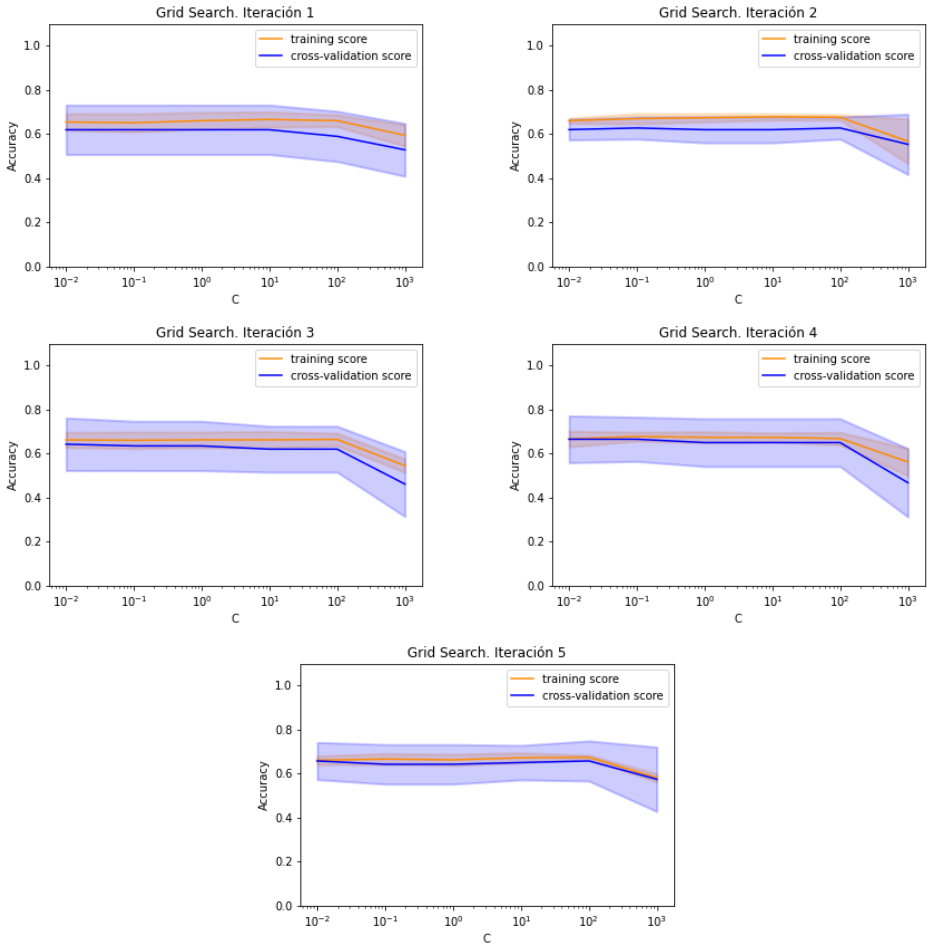


Figura 43. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características GRLab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	62,74	60,61
2	0,01	65,04	60,61
3	10	65,1	63,64
4	0,1	68,09	54,55
5	0,01	65,81	63,64
Media		65,36	60,61
Desviación típica		1,71	3,32

Tabla 13. Valores obtenidos del hiperparámetro C para Máquina de Soporte Vectorial Lineal. Vector de características GRL.

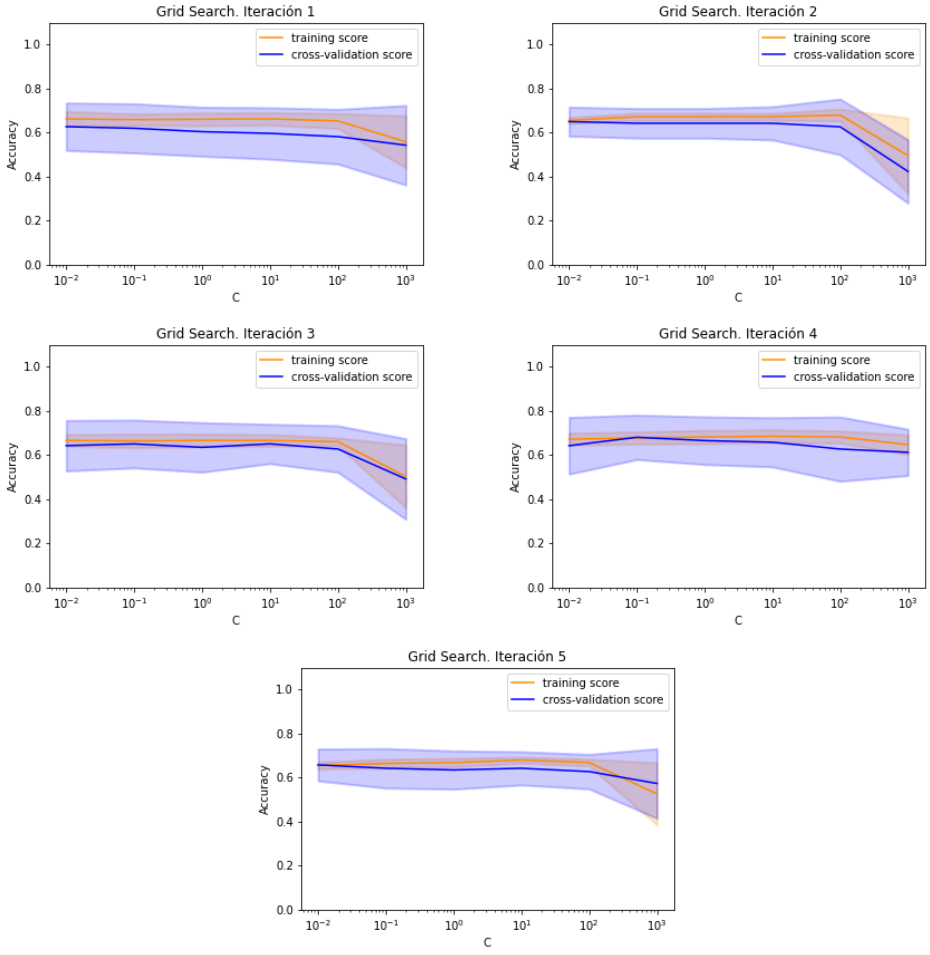


Figura 44. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Máquina de Soporte Vectorial Lineal con vector de características GRL.

6.3.2.2 Máquina de Soporte Vectorial: kernel RBF

Como análisis preliminar se ajustó únicamente el hiperparámetro γ. Al igual que en el apartado anterior, se consideró que los resultados fueron uniformes, dado que el valor ajustado de γ era el mismo en casi todas las iteraciones. Al visualizar la evolución de la precisión media de los modelos de entrenamiento y validación con cada valor de γ (figuras 45 a 48), se observó como a partir del de determinado valor la precisión del modelo de entrenamiento era muy superior a la del modelo de validación. Esto indica que el modelo se ajustaba muy bien a los datos de entrenamiento, pero no era capaz de generalizar con los datos de validación; es decir, se para valores de gamma superiores a 0,1 el modelo se sobreajustaba.

Nº iteración	γ	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	57,44	54,55
2	0,1	59,74	63,64
3	0,1	63,45	54,55
4	0,1	56,64	54,55
5	0,1	61,25	54,55
Media		59,70	56,37
Desviación típica		2,49	3,64

Tabla 14. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab Inicial.

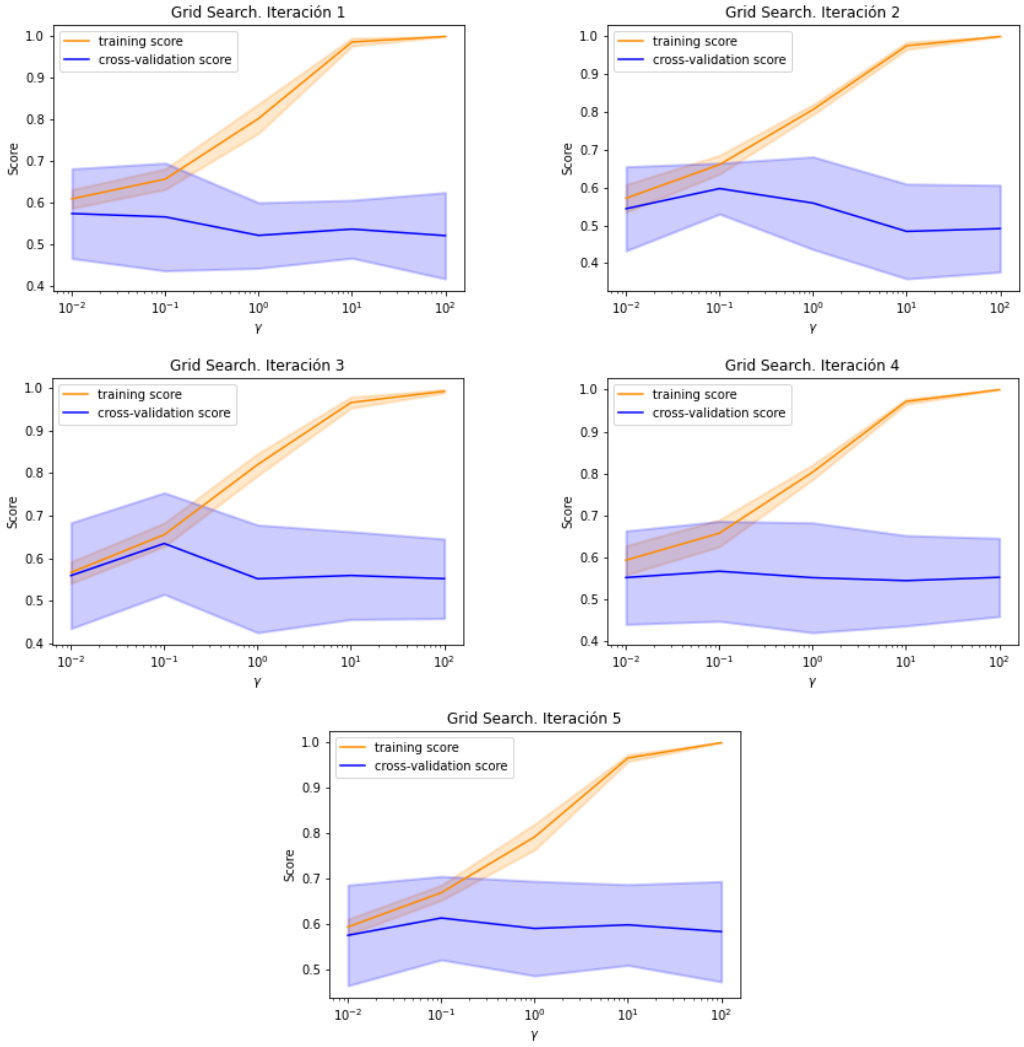


Figura 45. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características HSab Inicial.

Nº iteración	γ	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	63,5	60,61
2	10	65,07	57,58
3	0,1	65,81	66,67
4	1	65,1	69,7
5	0,1	65,84	60,61
Media		65,06	63,03
Desviación típica		0,85	4,45

Tabla 15. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab.

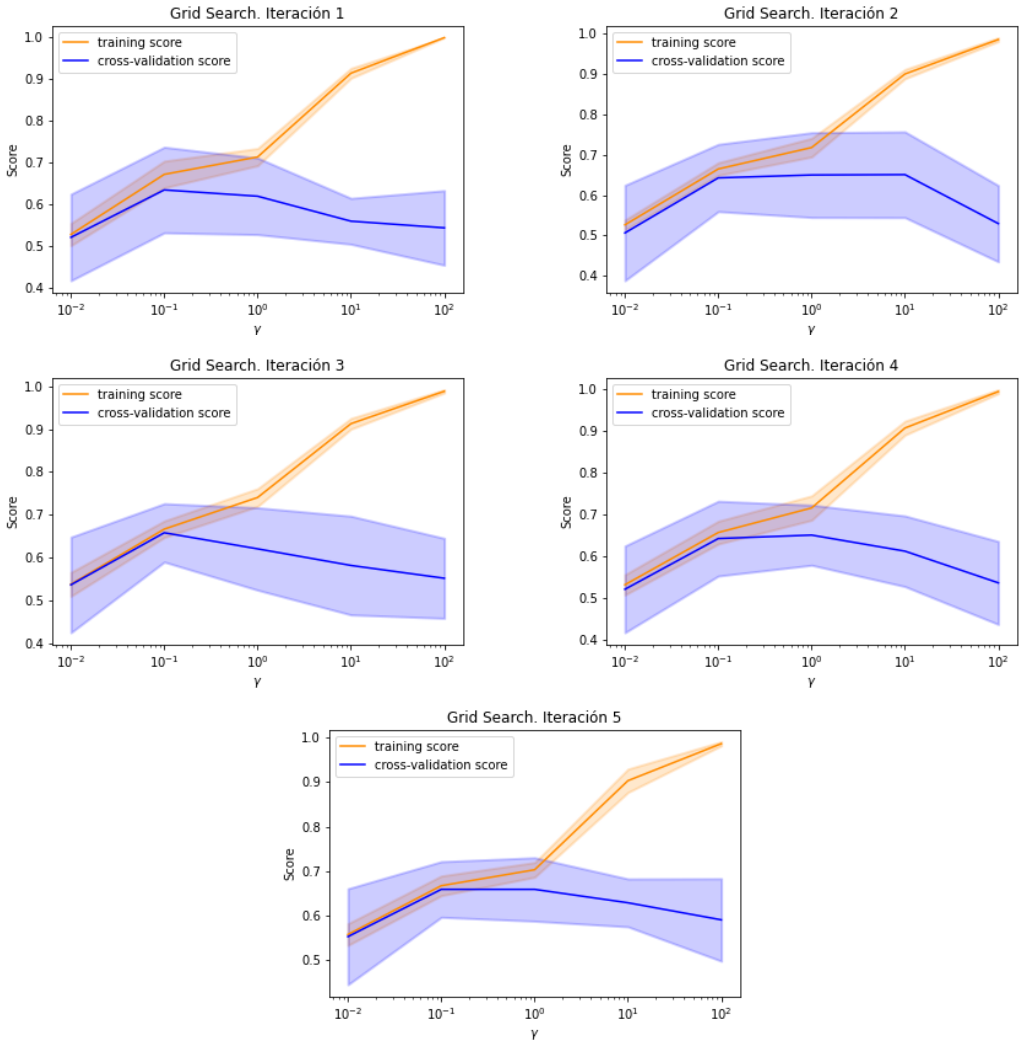


Figura 46. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características HSab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	64,25	60,61
2	1	65,01	66,67
3	1	63,69	54,55
4	0,1	65,01	63,64
5	0,1	64,27	66,67
Media		64,45	62,43
Desviación típica		0,51	4,53

Tabla 16. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRLab.

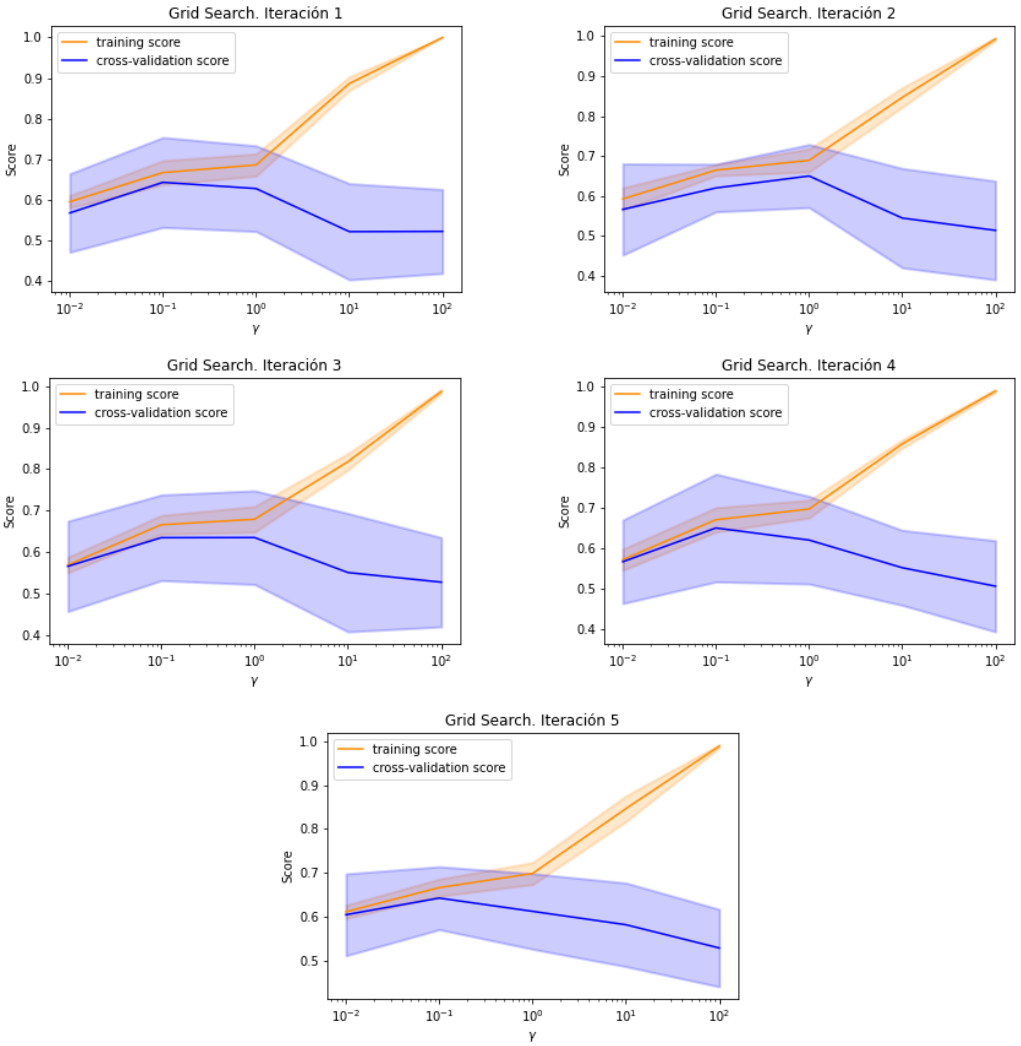


Figura 47. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características GRLab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	61,97	63,64
2	1	65,01	63,64
3	1	64,33	51,52
4	0,1	65,01	60,61
5	0,1	64,27	57,58
Media		64,12	59,40
Desviación típica		1,12	4,53

Tabla 17. Valores obtenidos del hiperparámetro γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRL.

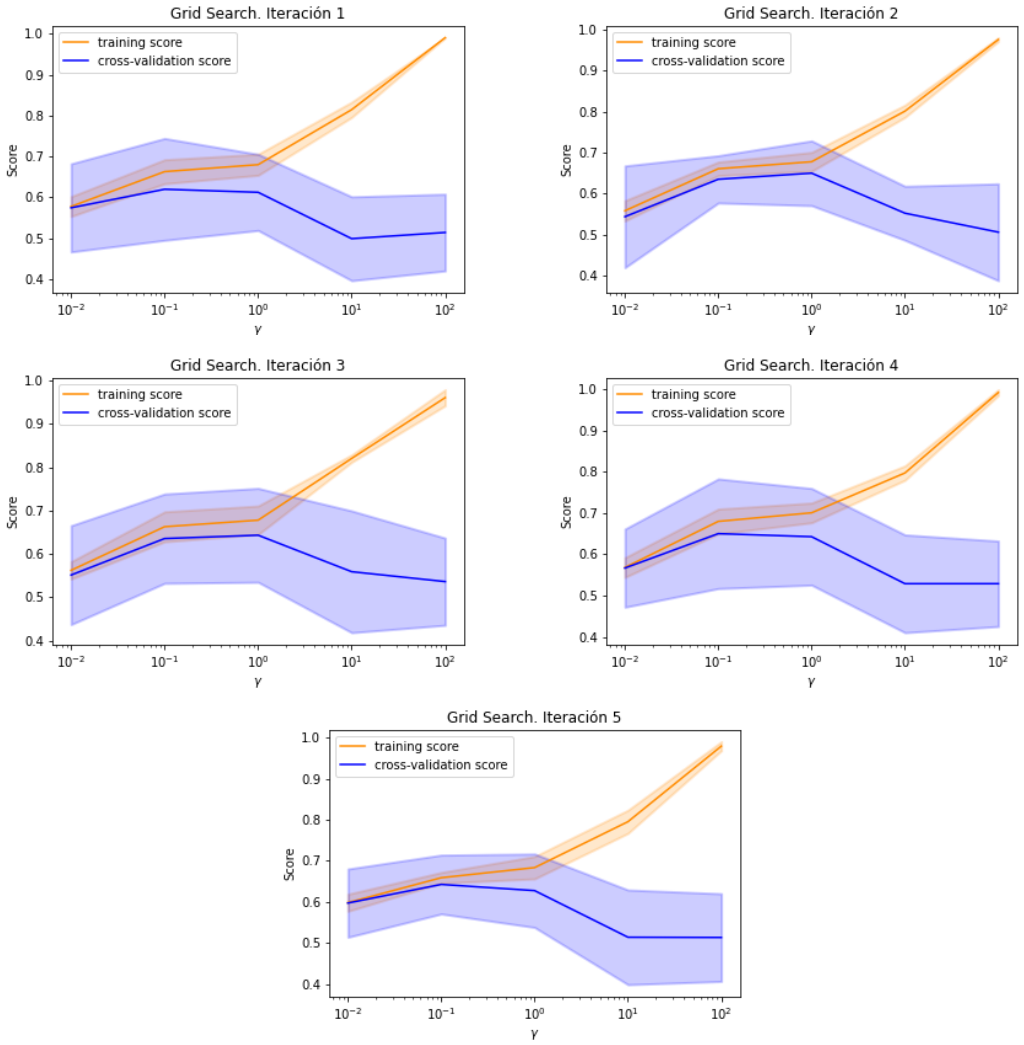


Figura 48. Visualización de la precisión media de entrenamiento y validación para cada valor de γ. Máquina de Soporte Vectorial RBF con vector de características GRL.

Tras observar el comportamiento del modelo ajustando un único hiperparámetro, se repitió el mismo procedimiento ajustando esta vez *C* y γ. Esta vez, los resultados mostraron una menor generalización en los vectores de características HSab (tabla 19), GRLab (tabla 20) y GRL (tabla 21) al no repetirse ninguna combinación de parámetros en la mayoría de las iteraciones. Para el vector de características HSab Inicial (tabla 18), se obtuvieron resultados más uniformes, pero con un menor nivel de predicción en el modelo de validación.

Nº iteración	γ	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	100	61,25	61,25
2	0,01	100	62,76	62,76
3	0,1	1	63,45	63,45
4	0,01	100	61,23	61,23
5	0,1	10	66,61	66,61
Media			63,06	63,06
Desviación típica			1,97	1,97

Tabla 18. Valores obtenidos de los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab Inicial.

Nº iteración	γ	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	10	65,04	65,04
2	10	10	65,16	65,16
3	0,1	1	65,81	65,81
4	0,01	1000	65,90	65,90
5	0,1	10	66,61	66,61
Media			65,70	65,70
Desviación típica			0,57	0,57

Tabla 19. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características HSab.

Nº iteración	γ	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	0,1	64,25	64,25
2	1	1	65,01	65,01
3	0,1	10	65,01	65,01
4	0,01	10	65,04	65,04
5	0,01	10	65,81	65,81
Media			65,02	65,02
Desviación típica			0,49	0,49

Tabla 20. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRLab.

Nº iteración	γ	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	100	61,99	61,99
2	1	1	65,10	65,10
3	0,1	10	65,10	65,10
4	0,01	100	65,54	65,54
5	0,01	10	65,81	65,81
Media			64,71	64,71
Desviación típica			1,39	1,39

Tabla 21. Valores obtenidos para los hiperparámetros C y γ para Máquina de Soporte Vectorial con kernel RBF. Vector de características GRL.

6.3.2.3 Máquina de Soporte Vectorial: kernel polinómico

En la máquina de soporte vectorial con *kernel* polinómico, al igual que en la de *kernel* lineal, se comenzó el análisis ajustando un único hiperparámetro, el grado del polinomio (*d* o *degree*). Respecto al ajuste del hiperparámetro *d*, se obtuvieron buenos resultados para los vectores de características GRLab (tabla 24) y GRL (tabla 25) puesto que en todas las iteraciones se estableció el mismo valor óptimo. El vector de características HSab inicial (tabla 22) generó resultados menos homogéneos, mientras que el vector de características HSab

(tabla 23) produjo resultados muy dispares, sin coincidir ningún valor en la mayoría de las iteraciones.

Por otro lado, al representar las precisiones alcanzadas con los subconjuntos de entrenamiento y validación para cada valor de d, se pudo observar un claro sobreajuste del modelo en los vectores de características HSab Inicial (figura 49) y HSab (figura 50) con *kernel*s polinómicos de grado superior a 2. Con en los vectores de características GRLab (figura 51) y GRL (figura 52) el modelo se ajustó mejor a los datos de entrenamiento que a los de validación, pero no se produjo sobreajuste.

Nº iteración	d	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	57,44	51,52
2	5	57,41	60,61
3	5	58,15	69,70
4	2	56,64	57,58
5	5	60,51	48,48
Media		58,03	57,58
Desviación típica		1,33	7,42

Tabla 22. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab Inicial.

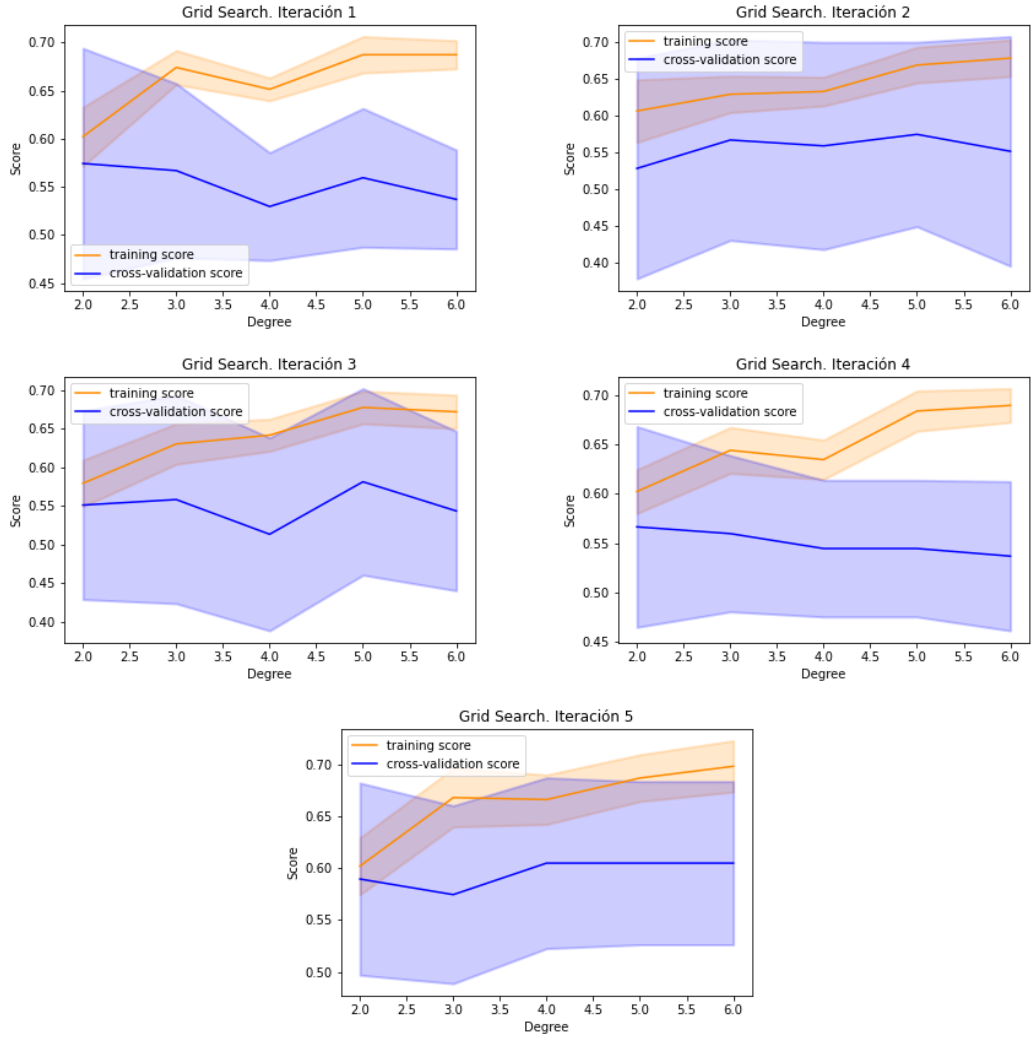


Figura 49. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características HSab Inicial.

Nº iteración	d	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	62,74	57,58
2	5	61,25	66,67
3	6	60,46	66,67
4	3	61,97	66,67
5	3	62,79	57,58
Media		61,84	63,03
Desviación típica		0,89	4,45

Tabla 23. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab.

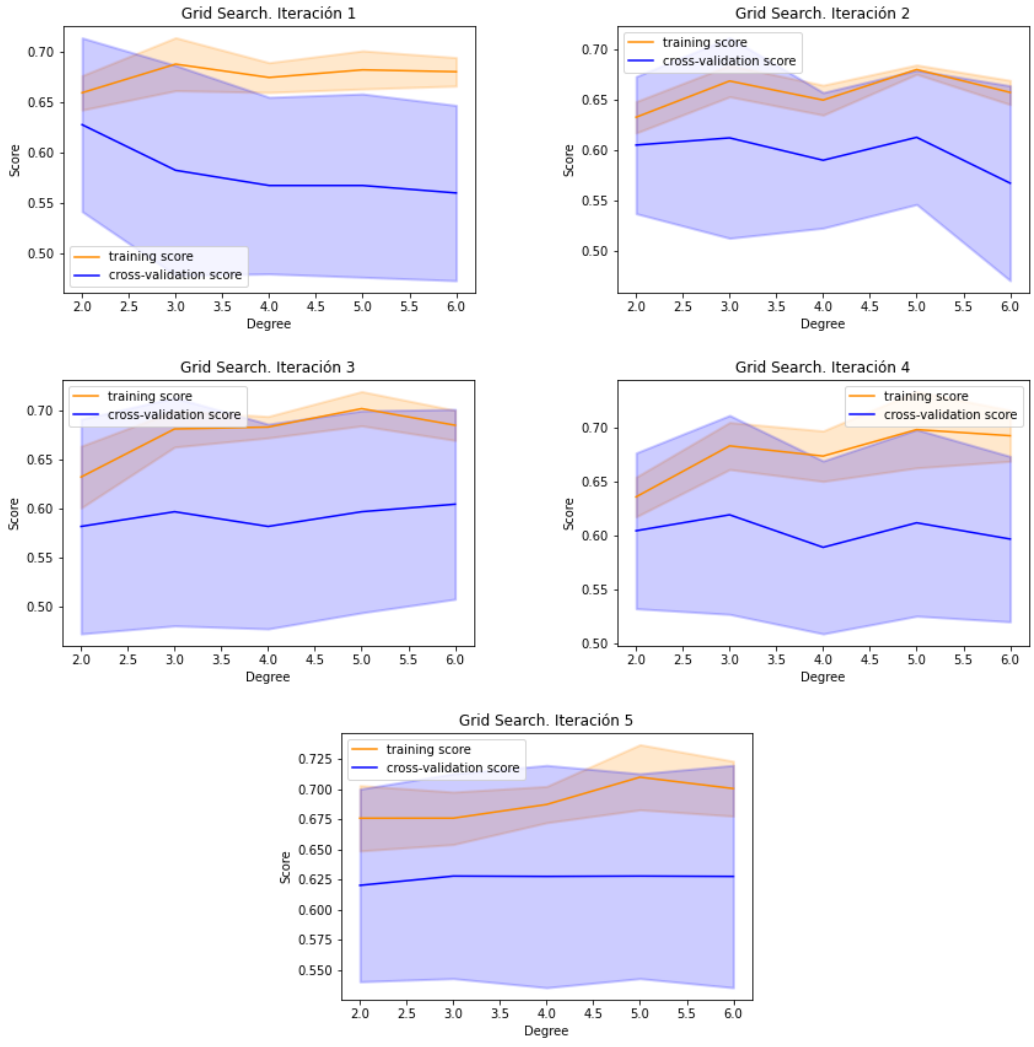


Figura 50. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características HSab.

Nº iteración	d	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	63,48	60,61
2	2	63,53	66,67
3	2	63,53	63,64
4	2	58,95	66,67
5	2	64,27	51,52
Media		62,75	61,82
Desviación típica		1,92	5,62

Tabla 24. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRLab.

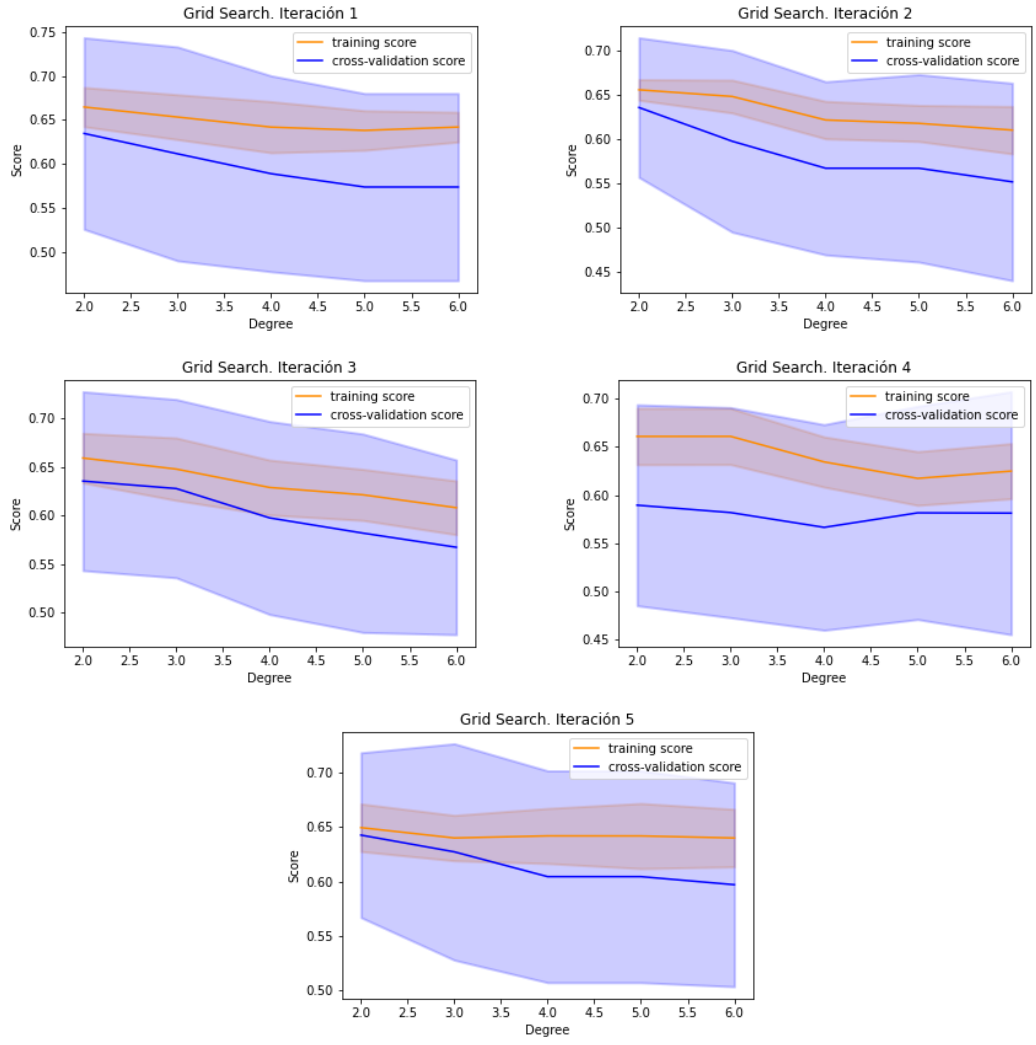


Figura 51. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características GRLab.

Nº iteración	d	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	63,45	60,61
2	2	61,99	66,67
3	2	65,07	63,64
4	2	57,44	66,67
5	2	63,50	48,48
Media		62,29	61,21
Desviación típica		2,61	6,75

Tabla 25. Valores obtenidos del hiperparámetro d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRL.

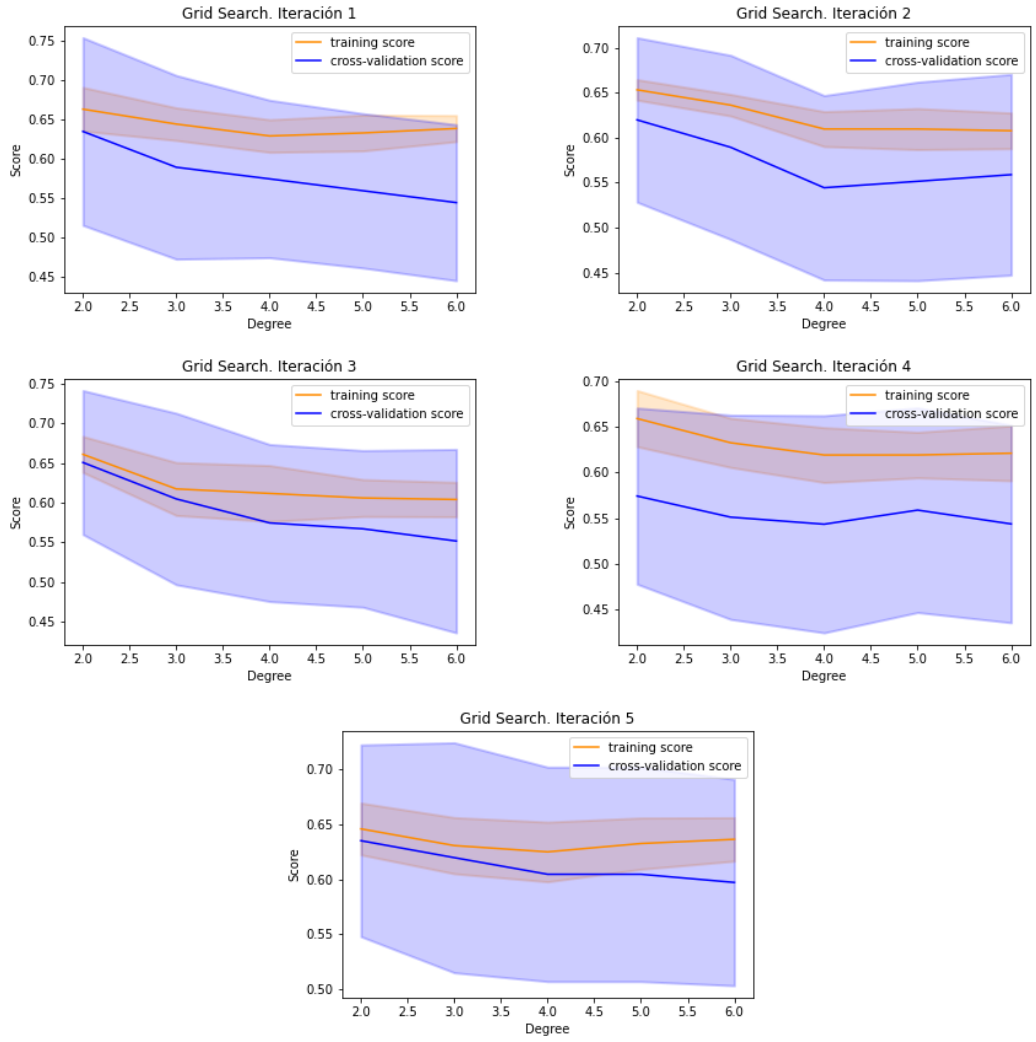


Figura 52. Visualización de la precisión media de entrenamiento y validación para cada valor de d. Máquina de Soporte Vectorial polinómica con vector de características GRL.

Después de este análisis previo, se procedió a repetir el procedimiento ajustando ls hiperparámetros C y d. Esta vez los resultados fueron menos dispersos que en el ajuste anterior, pero solo los vectores de características GRLab (tabla) y GRL (tabla) produjeron la misma combinación de valores en la mayoría de sus iteraciones. En cuanto a la precisión del clasificador, se mantiene por debajo del 70%.

Nº iteración	d	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	5	100	57,49	57,49
2	3	10	61,28	61,28
3	5	10	58,18	58,18
4	5	100	62,71	62,71
5	3	100	62,19	62,19
Media			60,37	60,37
Desviación típica			2,13	2,13

Tabla 26. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab Inicial.

Nº iteración	d	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	1	62,74	62,74
2	3	100	64,25	64,25
3	4	10	63,5	63,5
4	2	10	61,99	61,99
5	2	10	62,79	62,79
Media			63,05	63,05
Desviación típica			0,77	0,77

Tabla 27. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características HSab.

Nº iteración	d	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	1	63,48	63,48
2	2	10	65,04	65,04
3	2	1	63,53	63,53
4	3	10	61,28	61,28
5	2	1	64,27	64,27
Media			63,52	63,52
Desviación típica			1,26	1,26

Tabla 28. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRLab.

Nº iteración	d	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	2	1	63,45	63,45
2	2	10	62,79	62,79
3	2	10	65,81	65,81
4	2	10	61,25	61,25
5	2	1000	64,33	64,33
Media			63,53	63,53
Desviación típica			1,52	1,52

Tabla 29. Valores obtenidos para los hiperparámetros C y d para Máquina de Soporte Vectorial con kernel polinómico. Vector de características GRL.

6.3.2.4 Clasificador Random Forest

Como se expuso en el apartado 5.1.2, el ajuste de los parámetros de este clasificador se realizó en dos partes. En primer lugar, se probó una amplia gama de valores para los múltiples hiperparámetros mediante un algoritmo que seleccionaba en cada iteración una combinación diferente de características de entre todas las posibles. Para determinar si la búsqueda aleatoria produjo un mejor modelo, se comparó el modelo base con el mejor modelo obtenido. La tabla muestra los parámetros del modelo por defecto.

n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	bootstrap
100	10	4	'auto'	50	True

Tabla 30. Valores por defecto de los hiperparámetros del clasificador Random Forest.

El vector de características HSab Inicial (tablas 31 y 32) produjo resultados poco homogéneos, presentando poca homogeneidad en los valores obtenidos para cada hiperparámetro en las distintas iteraciones. El vector de características GRLab (tablas 35 y 36) dio resultados similares en la mayoría de las iteraciones para casi todos los hiperparámetros, pero fueron los vectores de características GRL (tablas 37 y 38) y HSAB (tablas 33 y 34) los que proporcionaron los resultados más homogéneos. A partir de las soluciones obtenidas con estos dos últimos vectores, se redujo el rango de cada hiperparámetro y se llevó a cabo una búsqueda de rejilla. En la tabla 39 se muestran los nuevos rangos establecidos para cada hiperparámetro y vector de características.

Parámetro	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5
n_estimators	400	400	2000	600	2000
min_samples_split	5	2	5	5	10
min_samples_leaf	1	4	1	2	2
max_features	'sqrt'	'sqrt'	'sqrt'	'auto'	'auto'
max_depth	30	10	10	70	40
bootstrap	True	True	True	False	True

Tabla 31. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características HSab Inicial.

Precisión	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Media	Desviación típica
Precisión entrenamiento (%)	62,88	60,61	61,36	63,64	69,7	63,64	3,21
Precisión modelo base (%)	63,64	63,64	63,64	72,73	39,39	60,61	11,18
Precisión validación (%)	63,64	66,67	66,67	66,67	33,33	59,40	13,09

Tabla 32. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características HSab Inicial.

Parámetro	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5
n_estimators	1400	1600	2000	1600	1400
min_samples_split	10	2	2	2	2
min_samples_leaf	2	4	2	4	1
max_features	'sqrt'	'auto'	'auto'	'sqrt'	'auto'
max_depth	80	10	90	10	100
bootstrap	True	True	True	True	True

Tabla 33. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características HSab.

Precisión	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Media	Desviación típica
Precisión entrenamiento (%)	67,42	70,45	65,91	65,15	67,42	67,27	1,82
Precisión modelo base (%)	63,64	63,64	66,67	69,7	63,64	65,46	2,42
Precisión validación (%)	66,67	63,64	63,64	69,7	66,67	66,06	2,27

Tabla 34. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características HSab.

Parámetro	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5
n_estimators	400	1000	200	800	1600
min_samples_split	10	5	5	10	2
min_samples_leaf	4	4	4	4	4
max_features	'auto'	'auto'	'auto'	'sqrt'	'sqrt'
max_depth	70	100	10	10	10
bootstrap	True	True	True	True	True

Tabla 35. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características GRLab.

Precisión	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Media	Desviación típica
Precisión entrenamiento (%)	61,36	64,39	63,64	59,89	63,64	62,58	1,69
Precisión modelo base (%)	63,64	57,58	60,61	57,58	57,58	59,40	2,42
Precisión validación (%)	63,64	60,61	54,55	60,61	60,61	60,00	2,97

Tabla 36. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características GRLab.

Parámetro	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5
n_estimators	800	400	800	400	1000
min_samples_split	10	10	10	10	10
min_samples_leaf	4	4	4	4	4
max_features	'sqrt'	'auto'	'sqrt'	'auto'	'auto'
max_depth	50	70	50	70	50
bootstrap	True	True	True	True	True

Tabla 37. Valores obtenidos para los hiperparámetros del clasificador Random Forest. Vector de características GRL.

Precisión	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Media	Desviación típica
Precisión entrenamiento (%)	60,61	62,88	63,64	62,12	66,67	63,18	2,01
Precisión modelo base (%)	54,55	57,58	57,58	63,64	51,52	56,97	4,02
Precisión validación (%)	63,64	63,64	57,58	60,61	51,52	59,40	4,53

Tabla 38. Precisión del modelo durante el ajuste de hiperparámetros del clasificador Random Forest. Vector de características GRL.

Hiperparámetro	Valores asignados HSab					Valores asignados GRL			
n_estimators	1400	1500	1600	2000		100	400	800	1000
max_depth	10	50	80	90	100	50	70	None	
min_samples_split	2	5	10			2	5	10	
min_samples_leaf	1	2	4			1	4		
max_features	'auto'	'sqrt'				'auto'	'sqrt'		
bootstrap	True					True			

Tabla 39. Valores asignados a los hiperparámetros del clasificador Random Forest en la búsqueda de rejilla

Mediante la búsqueda de rejilla se obtuvieron los modelos óptimos para cada vector de características. Ambos modelos obtuvieron buenos resultados, con una precisión de validación superior a la de entrenamiento (tablas 40 y 41).

Vector de características	n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	bootstrap
GRL	100	10	4	'auto'	50	True
Hsab	1400	2	1	'auto'	10	True

Tabla 40. Modelos óptimos obtenidos para el clasificador Random Forest con los vectores de características HSab y GRL.

Vector de características	Precisión de entrenamiento (%)	Precisión de validación (%)
GRL	62,61	66,00
Hsab	62,61	68,00

Tabla 41. Precisiones de entrenamiento y validación obtenidas con el modelo óptimo del clasificador Random Forest para los vectores de características HSab y GRL.

6.3.2.5 Regresión logística

Para todos los vectores de características estudiados, se obtuvo el mismo valor óptimo para el hiperparámetro *C* en la mayoría de las iteraciones, mostrando un buen nivel de generalización en el ajuste.

En el caso de los vectores de características HSab Inicial (tabla 42) y HSab (tabla 43) la precisión obtenida para el modelo de validación fue muy baja y poco constante. Además, fue inferior a la precisión del modelo de entrenamiento, que si se mantuvo homogénea en todas las iteraciones. Por este motivo, no se consideraron uno vectores de entrada adecuados para este clasificador y se descartaron como opciones para clasificador óptimo.

Por otro lado, los vectores de características GRLab (tabla 44) y GRL (tabla 45) generaron resultados homogéneos tanto en la obtención del valor óptimo del hiperparámetro como en la evaluación de los modelos de entrenamiento y validación. Es por esto por lo que se consideraron los modelos obtenidos del ajuste con estos dos vectores como opción para el clasificador óptimo.

Al visualizar la evolución de las presiones de entrenamiento y validación en función del valor del hiperparámetro *C* en cada iteración (figuras 53 a 56) se comprobó que no se estaba produciendo sobreajuste, dado que ambos modelos (entrenamiento y validación) mantenían precisiones similares.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,1	62,05	48,48
2	0,01	59,03	39,39
3	0,01	59,00	57,58
4	0,01	61,20	51,52
5	0,01	63,59	42,42
Media		60,97	47,88
Desviación típica		1,77	6,47

Tabla 42. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características HSab Inicial.

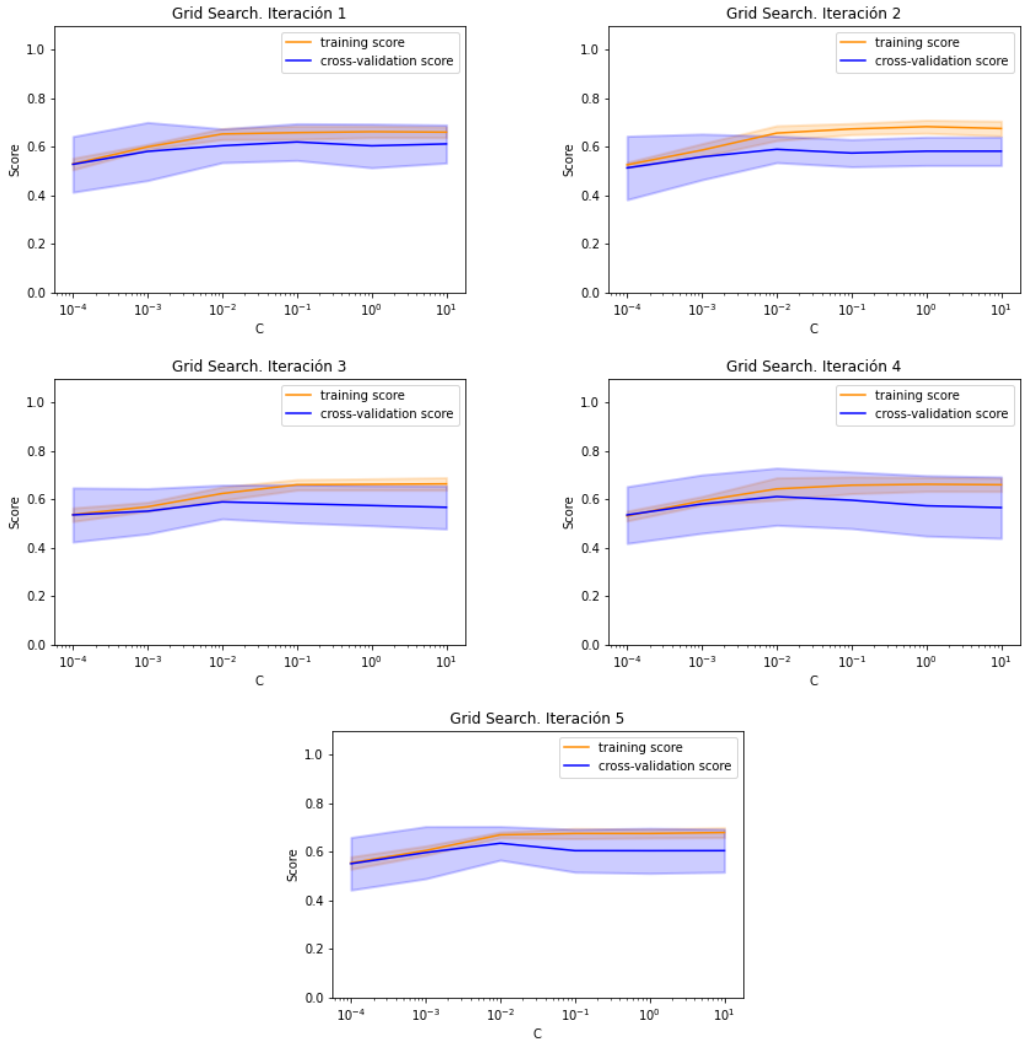


Figura 53. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características HSab Inicial.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	10	65,04	36,36
2	0,01	61,28	54,55
3	0,01	64,3	36,36
4	0,01	65,04	63,64
5	1	65,81	54,55
Media		64,29	49,09
Desviación típica		1,58	10,91

Tabla 43. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características HSab.

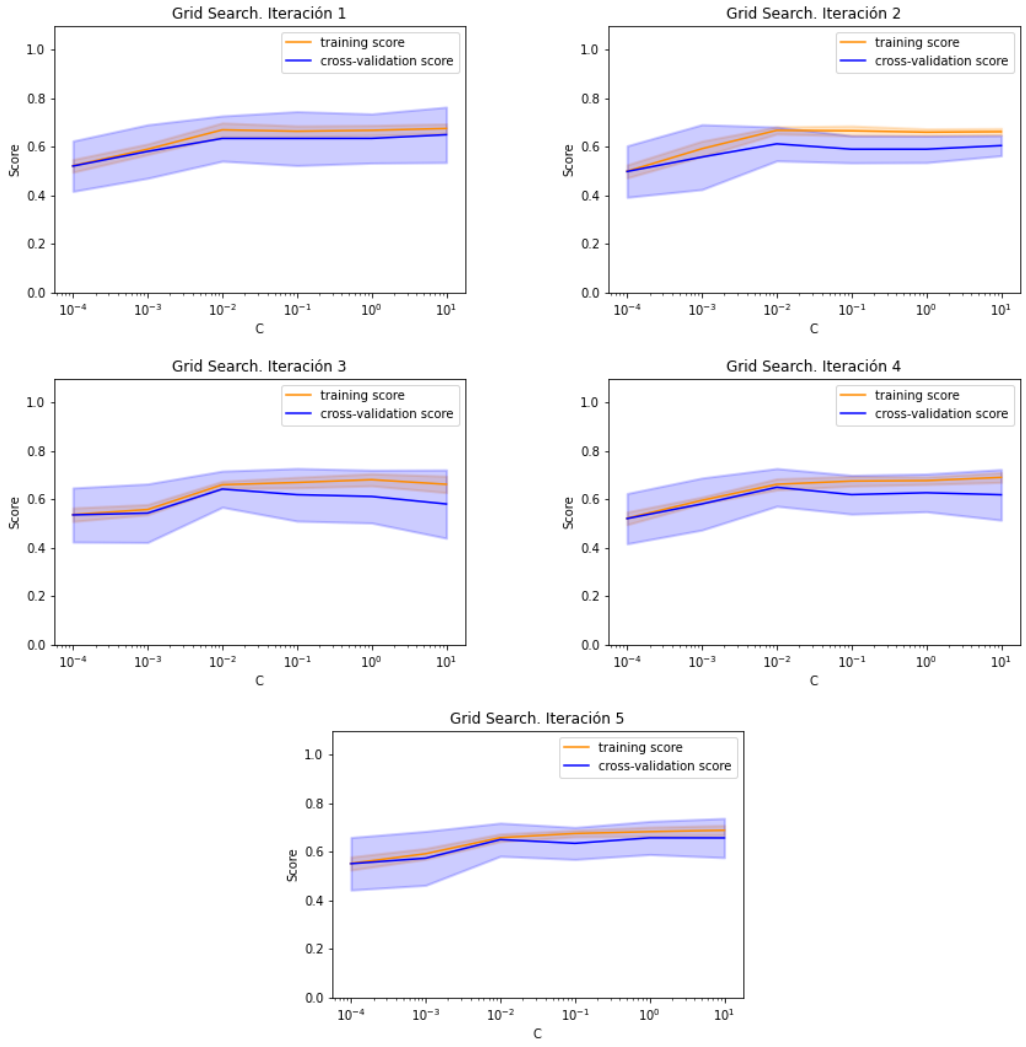


Figura 54. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características HSab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	10	62,82	66,67
2	0,01	64,3	63,64
3	0,01	64,3	66,67
4	0,01	65,07	69,70
5	10	67,38	66,67
Media		64,77	66,67
Desviación típica		1,49	1,92

Tabla 44. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características GRLab.

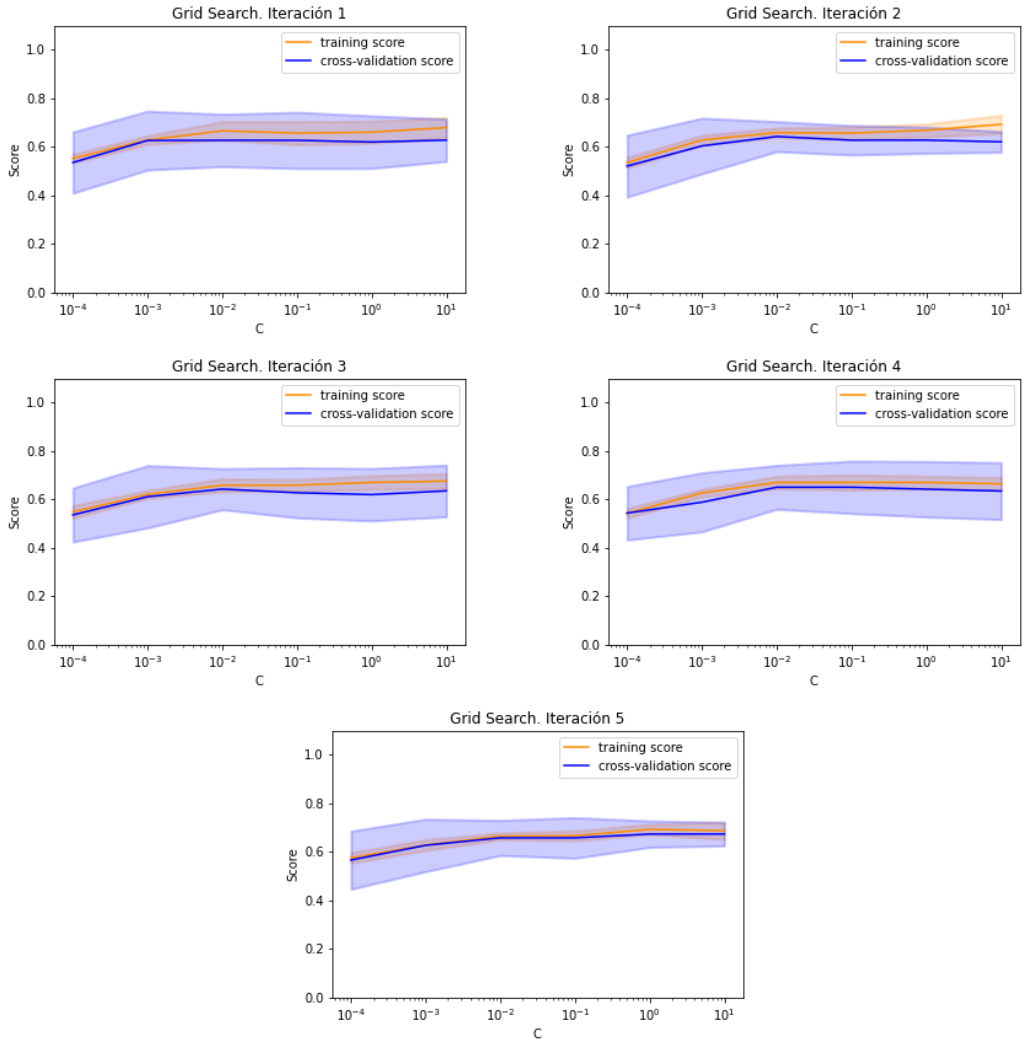


Figura 55. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características GRLab.

Nº iteración	C	Precisión Entrenamiento (%)	Precisión Validación (%)
1	0,01	62,74	66,67
2	0,01	62,79	54,55
3	0,01	65,84	60,61
4	0,01	65,81	60,61
5	10	66,61	69,70
Media		64,76	62,43
Desviación típica		1,65	5,28

Tabla 45. Valores obtenidos para el hiperparámetro C de la Regresión Logística. Vector de características GRL.

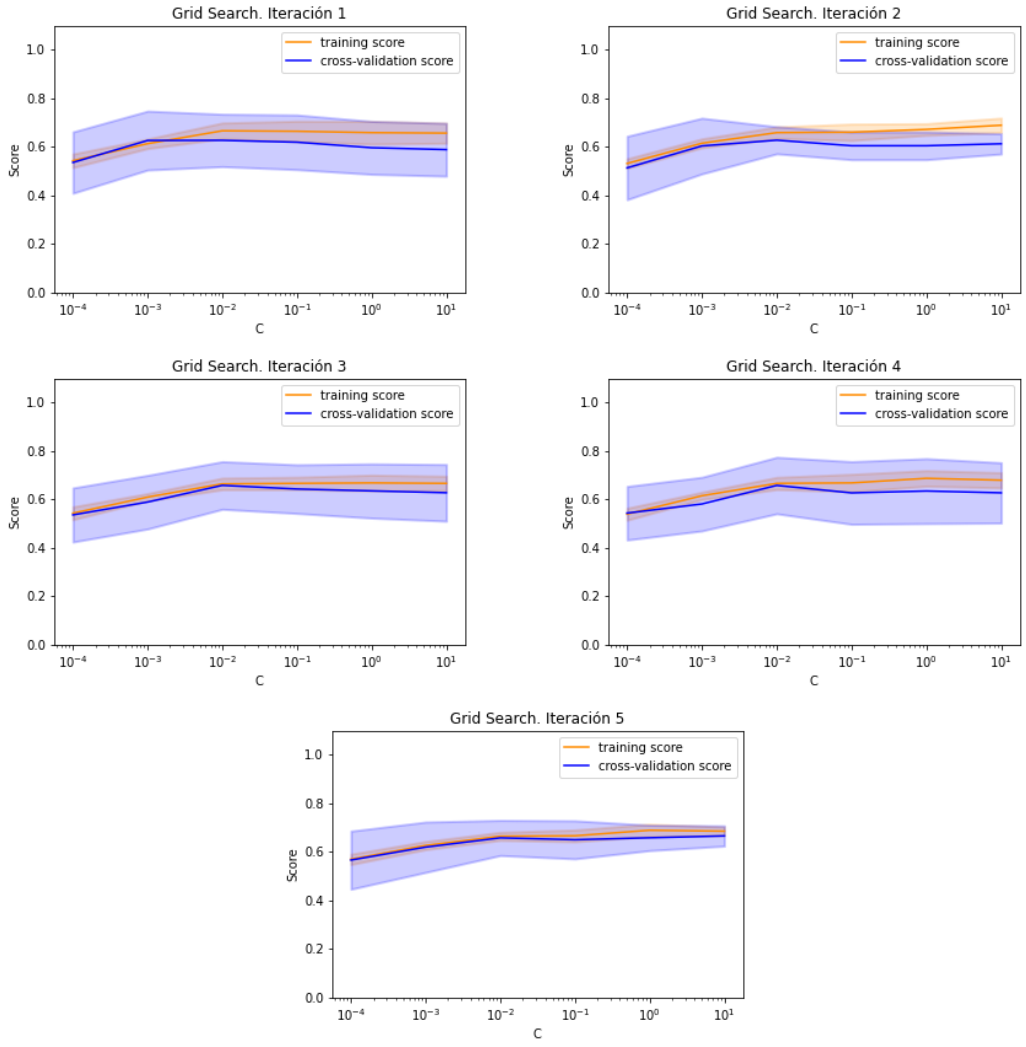


Figura 56. Visualización de la precisión media de entrenamiento y validación para cada valor de C. Regresión logística con vector de características GRL.

6.3.3 Clasificador óptimo

A partir de los resultados del apartado anterior, se consideraron varios modelos como posible clasificado óptimo. En la tabla 46 se muestran las características de estos modelos.

Clasificador	Vector de características	Hiperparámetros
SVM Lineal	GRLab	C = 0,01
	GRL	
SVM RBF	HSab incial	γ = 0,01 C = 100
	GRLab	γ = 0,01 C = 10
SVM polinómica	GRLab	d = 2 C = 1
	GRL	d = 2 C = 10
Random Forest	HSab	Configuración mostrada en la tabla 40
	GRL	
Regresión Logística	GRLab	C = 0,01
	GRL	

Tabla 46. Resumen de características de los clasificadores considerados como clasificador óptimo.

La tabla 47 resume los resultados obtenidos para los diferentes vectores de características con el clasificador correspondiente. Los resultados obtenidos fueron bastante similares para todos los modelos, aunque no del todo satisfactorios: todos los modelos presentan precisiones inferiores al 70% en sus predicciones.

Vector de características	Clasificador	Precisión media (%)	Desviación típica (%)
GRLab	SVM Lineal	64,24	3,53
GRL	SVM Lineal	63,64	1,92
GRLab	SVM con *kernel* RBF	64,85	1,48
HSab Inicial	SVM con *kernel* RBF	62,42	4,54
GRLab	SVM con *kernel* polinómico de grado 2	61,82	5,62
GRL	SVM con *kernel* polinómico de grado 2	62,42	5,62
HSab	Random Forest	65,45	4,54
GRL	Random Forest	58,79	2,42
GRLab	Regresión Logística Multinomial	63,64	5,07
GRL	Regresión Logística Multinomial	57,58	7,42

Tabla 47. Resumen de los resultados obtenidos.

Los mejores resultados los proporcionó la Máquina de Soporte Vectorial con *kernel* RBF para el vector de características GRLab, con una precisión del 64,85% en el subconjunto de prueba. Aunque el clasificador Random Forest obtuvo la precisión media más alta para el vector de características HSab (65,45%), su desviación típica fue mayor, lo que significa que hubo mayor variación en los resultados de cada una de las iteraciones de la validación cruzada, mostrando una peor capacidad de generalización del modelo.

Además, se analizaron los pesos de los modelos ajustados para las Máquinas de Soporte Vectorial lineales, el clasificador Random Forest y la regresión logística multinomial. La inspección de estas parcelas sugiere que la primera componente del PCA es la más discriminativa, ya que los coeficientes asociados a esta componente presentaron un valor superior a los demás.

Al inspeccionar los coeficientes asociados a las componentes para la Máquina de soporte Vectorial Lineal con los vectores GRLab (figura 57) y GRL (figura 58) se comprobó que estos pesos no se mantienen constantes en todas

las iteraciones y se notó una similitud entre los coeficientes arrojados con ambos vectores de características, aunque no es de extrañar teniendo en cuenta la similitud de estos dos vectores.

Por otro lado, los pesos asignados a las características en la función de decisión de la regresión logística mostraron formas similares con los dos vectores de características empleados. Sin embargo, los rangos de los valores asociados fueron diferentes y el vector de características GRLab (figura 59) mostró una mayor uniformidad que el vector de entrada GRL (figura 60), para el que se obtuvieron valores dispares en la tercera componente de PCA.

Por último, se analizó la importancia asignada a cada característica en los modelos con el clasificador Random Forest. Pudo comprobarse la robustez en la asignación de pesos a las componentes de PCA y se notó la similitud entre ambos modelos (figuras 61 y 62).

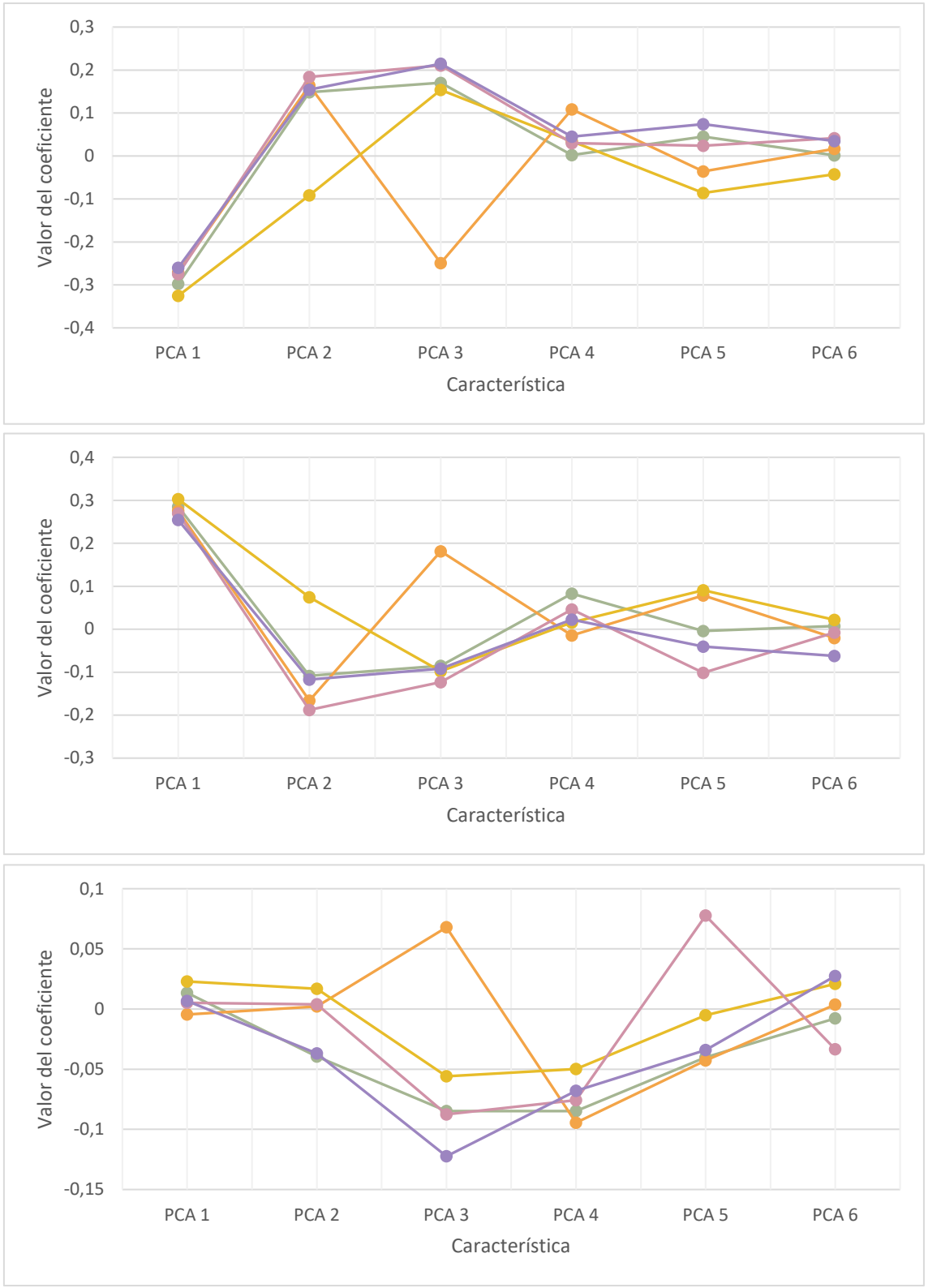


Figura 57. Coeficientes asociados con cada componente PCA para la Máquina de Soporte Vectorial Lineal con vector de características GRLab.

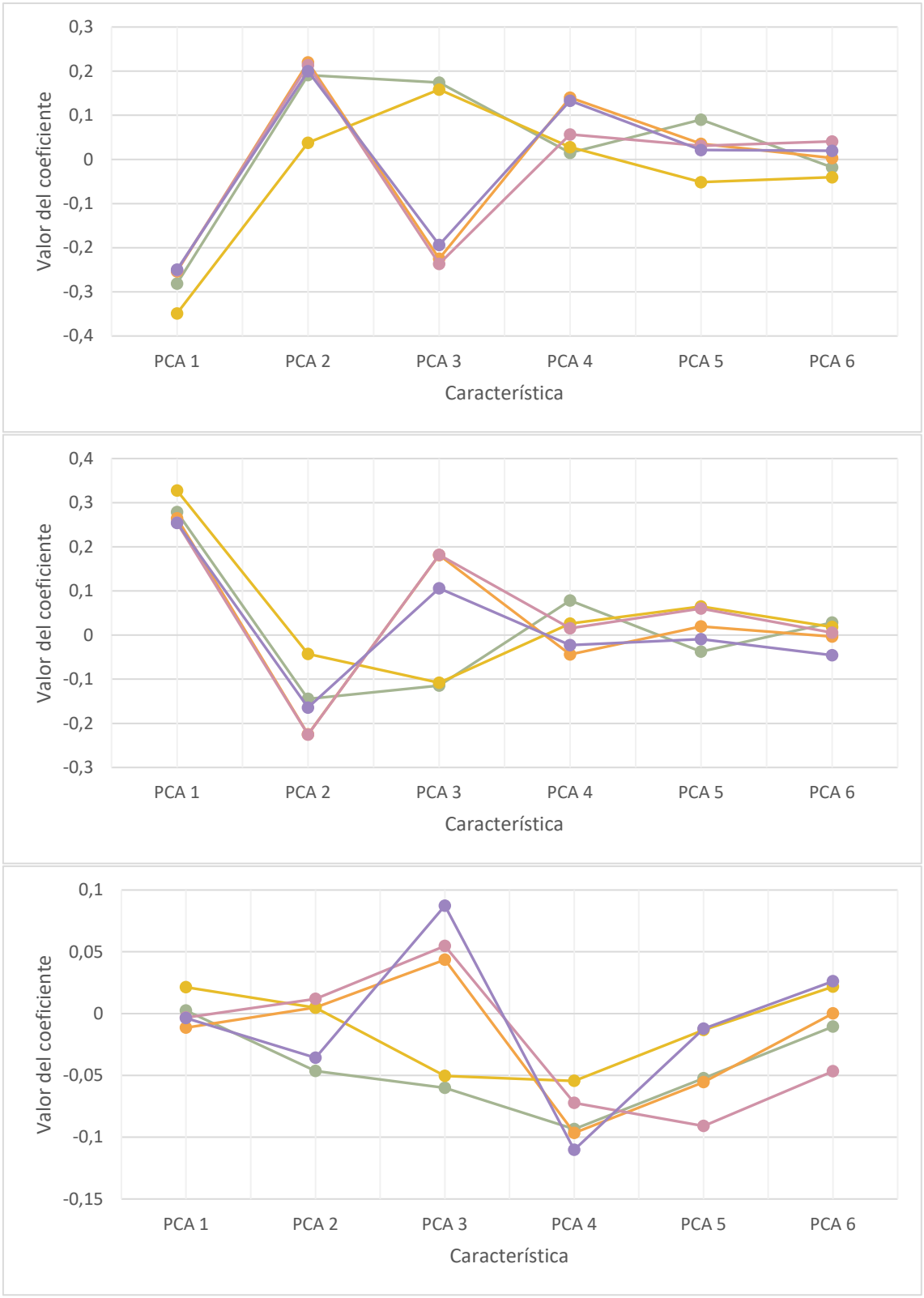


Figura 58. Coeficientes asociados con cada componente PCA para la Máquina de Soporte Vectorial Lineal con vector de características GRL.

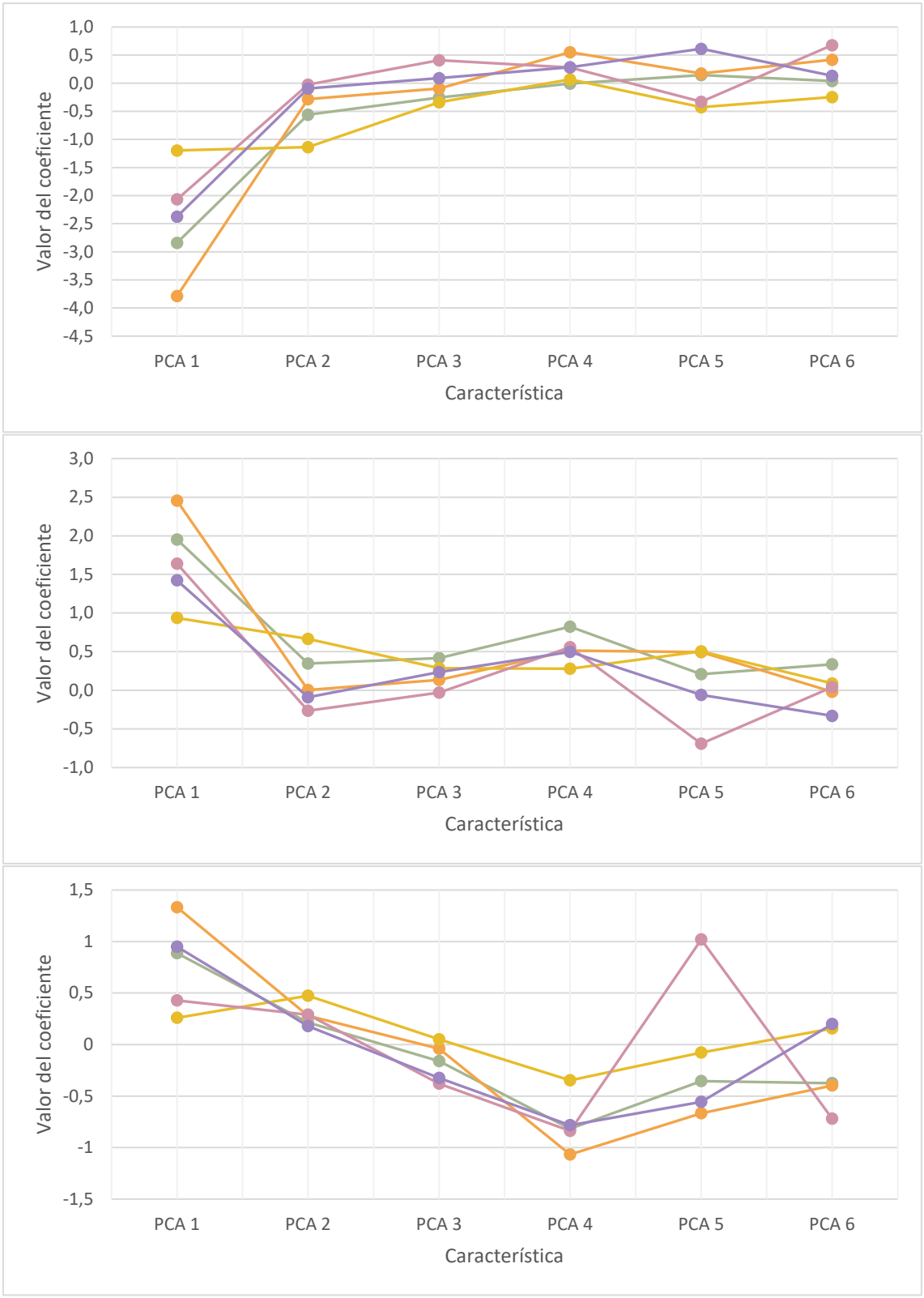


Figura 59. Coeficientes asociados con cada componente PCA para la Regresión Logística Multinomial con vector de características GRLab.

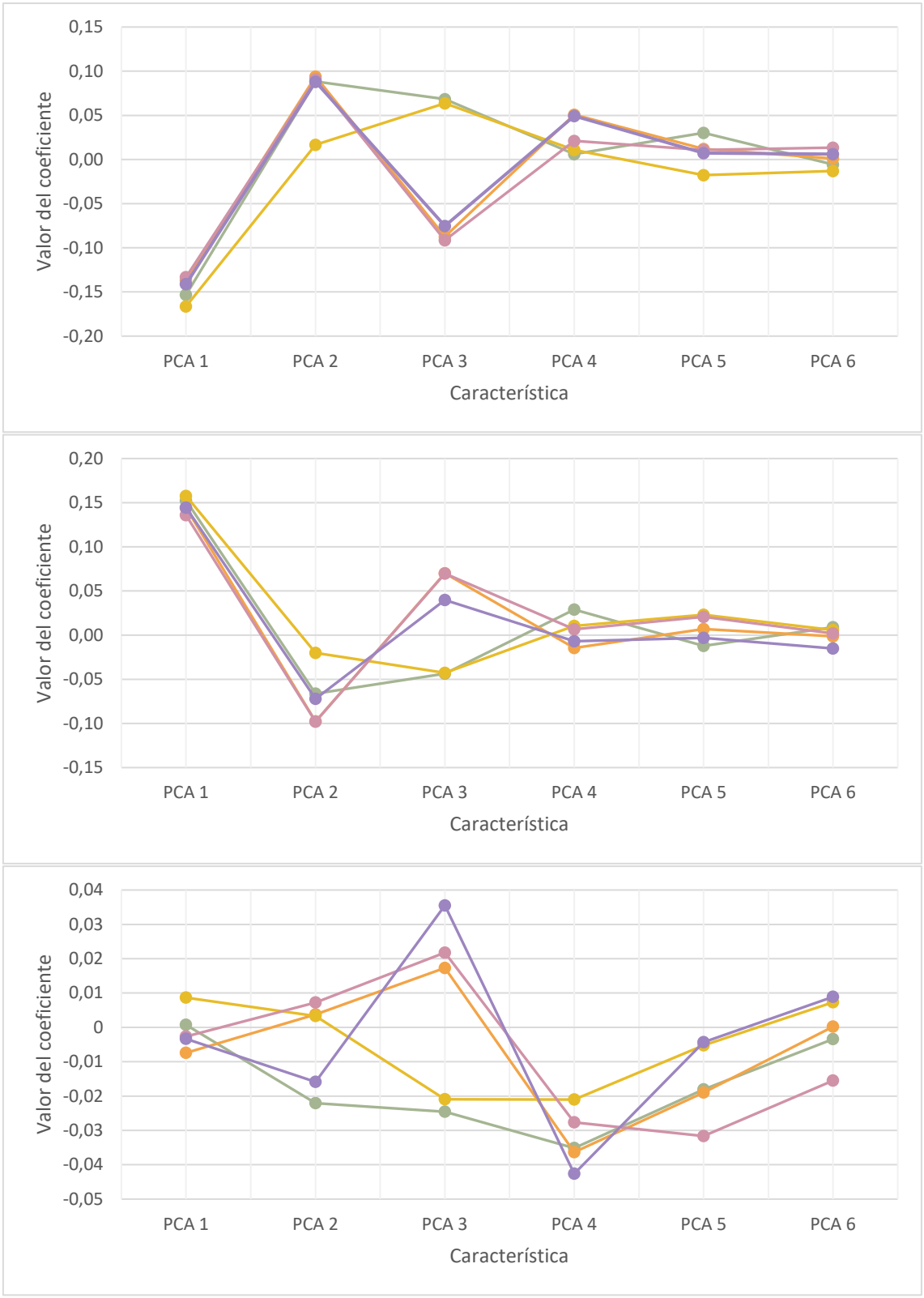


Figura 60. Coeficientes asociados con cada componente PCA para la Regresión Logística Multinomial con vector de características GRL.

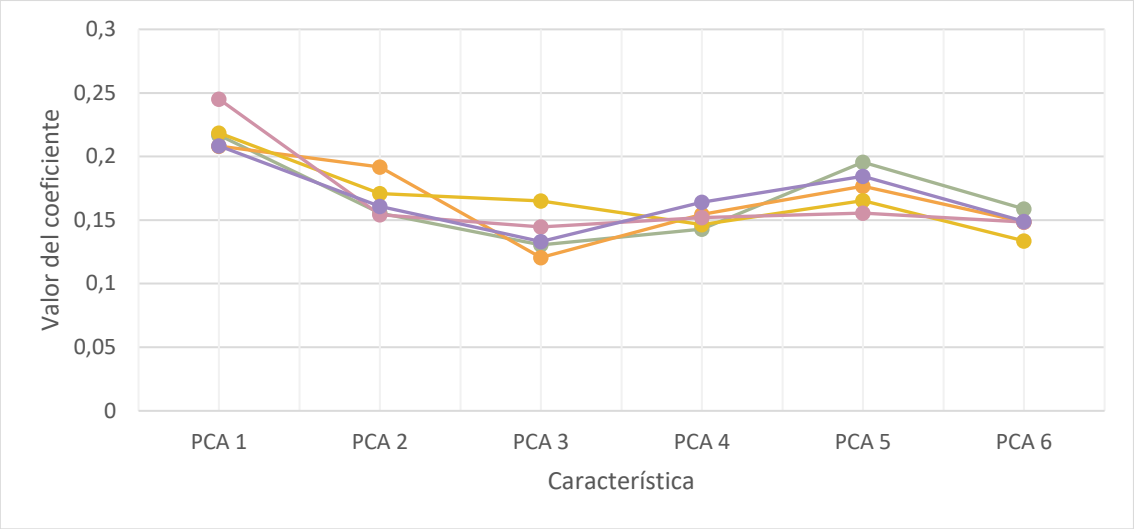


Figura 61. Coeficientes asociados con cada componente PCA para el clasificador Random Forest con vector de características HSab.

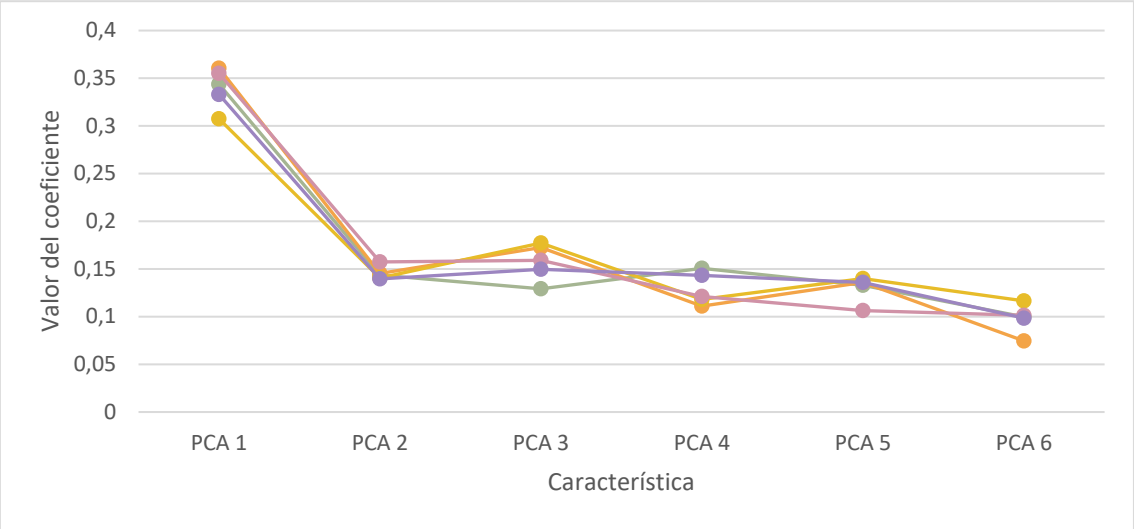


Figura 62. Coeficientes asociados con cada componente PCA para el clasificador Random Forest con vector de características GRL.

A la vista de los resultados expuestos, se aplicaron varias estrategias de mejora a dos de los modelos obtenidos con el fin de aumentar la precisión. Estos modelos fueron la Máquina de Soporte Vectorial con *kernel* RBF con el vector de características GRLab y el clasificador Random Forest con el vector de características HSab.

En primer lugar, sustituyó el iterador de validación cruzada K-fold por un iterador *LeaveOneOut* (LOO). Por otro lado, aumentó el número de componentes PCA y se empleó un método de extracción de características KPCA

Además, se realizó una evaluación heurística de los hiperparámetros tanto en los clasificadores como en el método de extracción de características para encontrar las combinaciones que aportaban una mayor precisión. En el caso del modelo Random Forest, para el método de extracción de características KPCA, se volvió a realizar la búsqueda de rejilla rangos establecidos para cada hiperparámetro mostrados en la tabla 39 para el vector de características HSab. El modelo obtenido se expone en la tabla 48. Como se puede comprobar, es muy similar al obtenido anteriormente para el mismo vector de características (tabla 48).

Vector de características	n_estimators	min_samples_split	min_samples_leaf	max_features	max_depth	bootstrap
Hsab	1600	10	1	'auto'	10	True

Tabla 48. Modelo óptimo obtenido para el clasificador Random Forest con el vector de características HSab y un método de extracción de características KPCA.

En la tabla 49 se exponen los resultados obtenidos. Se han producido ciertas mejoras en los resultados, sin embargo, todos los siguen presentando precisiones inferiores al 70% en sus predicciones.

Vector de características	Método de extracción de características	Nº características	Clasificador	Validación cruzada	Precisión media (%)	Desviación típica (%)
GRLab	PCA	6	SVM con kernel RBF	K-fold	64,85	1,48
GRLab	PCA	6	SVM con kernel RBF	LOO	66,06	-
GRLab	PCA	66	SVM con kernel RBF	K-fold	66,06	4,02
GRLab	KPCA	131	SVM con kernel RBF	K-fold	61,82	7,07
HSab	PCA	6	Random Forest	K-fold	65,45	4,54
HSab	PCA	132	Random Forest	K-fold	67,88	3,64
HSab	PCA	132	Random Forest	LOO	68,48	-
HSab	KPCA	131	Random Forest	K-fold	65,45	1,48

Tabla 49. Resumen de resultados en el estudio de mejora del clasificador óptimo.

El mejor resultado de clasificación se logró utilizando un método de extracción de características KPCA junto a un clasificador Random Forest con el vector de características HSab; con una precisión del 65,45% en el subconjunto de prueba. Aunque hubo modelos con una precisión media más elevada, su desviación típica fue mayor, lo que significa que hubo mayor variación en los resultados de cada una de las iteraciones de la validación cruzada, mostrando una peor capacidad de generalización del modelo.

Por otro lado, se comprobó que aplicar una estrategia de validación cruzada *LeaveOneOut* aumentaba la precisión alcanzada, sin embargo, no llegan a obtenerse los resultados esperados. Este método tiene un mayor coste computacional y tiene a sobreajustar el modelo, por lo que se prefiere la validación cruzada de tipo K-fold.

7 Interfaz gráfica de usuario

Como parte final del trabajo, se diseñó e implementó una interfaz gráfica de usuario (GUI o *Graphical User Interface*) que realiza una estimación del nivel de impurezas insolubles de una muestra de aceite de oliva a partir de su fotografía, empleando el clasificador óptimo obtenido en el apartado anterior.

En primer lugar, se elaboró un programa para entrenar el clasificador con todas las muestras disponibles. A partir de este programa, pudo exportarse tanto el clasificador entrenado como los métodos de escalado de datos y extracción de características ajustados empleando el módulo *pickle* de Python. Este módulo implementa protocolos binarios para serializar y deserializar una estructura de objeto Python (Python Software Foundation, 2021).

La interfaz gráfica se desarrolló mediante Streamlit, una biblioteca de Python de código abierto que facilita la creación y el intercambio de aplicaciones web personalizadas para el aprendizaje automático y la ciencia de datos (Streamlit, 2021).

En primer lugar, mediante una entrada de texto, la aplicación solicita al usuario la ubicación donde se almacenan el modelo entrenado y los métodos de escalado y extracción de características ajustados. Después, permite al usuario cargar una imagen, que muestra en la pantalla. Al presionar el botón “IDENTIFICAR”, se lleva a cabo la predicción.

Para realizar la predicción, primero es necesario cargar el modelo de clasificación entrenado, lo que se lleva a cabo mediante la función load_model, expuesta en el cuadro 7.

load_model (path)

Esta función carga el modelo entrenado.

- **path:** *str*. Ubicación del archivo.

Cuadro 7. Función que carga el modelo entrenado.

A continuación, se obtiene el vector de entrada a partir de la imagen cargada y se realiza el preprocesamiento de los datos aplicando los métodos de escalado y extracción de características. Para esto, se elaboraron las funciones auxilaires input_vector y image_preprocessing, cuyo propósito y parámetros se muestran en los cuadros 8 y 9.

input_vector (*image*)

Elabora el vector de características a partir de una imagen proporcionada.

- **image**: *array of uint8*. Imagen de muestra.

Cuadro 8. Función que elabora el vector de características de una imagen proporcionada.

image_preprocessing (*image, path*)

Preprocesamiento de los datos de entrada al clasificador. Emplea la función auxiliar input_vector para obtener el vector de características a partir de la imagen de muestra proporcionada.

- **image**: *array of uint8*. Imagen de muestra.
- **path:** *str*. Ubicación del archivo.

Cuadro 9. Función que realiza el preprocesamiento de los datos de entrada al clasificador.

El último paso, es realizar la estimación del contenido de impurezas insolubles y mostrar los resultados. Para ejecutar el clasificador, se desarrolló la función model_prediction (cuadro 10), que aplica el clasificador óptimo diseñado para clasificar el nivel de impurezas en bajo, medio o alto.

model_prediction (*X_r*, *clf*)

Realiza la predicción del nivel de impurezas de una muestra de aceite de oliva a partir de su vector de características y un clasificador dado.

- **X_r:** *array of float64*. Vector de características de la imagen de muestra.
- **clf:** *svm._classes.SVC*. Clasificador entrenado.

Cuadro 10. Función que devuelve la predicción del nivel de impurezas de una muestra de aceite de oliva

8 Discusión y conclusión

Tal y como se ha descrito en los epígrafes preliminares de este TFM, la necesidad de mantener la competitividad del sector del aceite de oliva en un entorno cada vez más exigente, obliga a explorar nuevos procedimientos para controlar la calidad de este producto como son los basados en la visión por computador. No obstante, aunque se trata de una línea de trabajo que está siendo desarrollada en distintos sectores agroalimentarios, en el sector olivarero la investigación publicada todavía es escasa en lo que se refiere a la detección de impurezas en la extracción del aceite. En este contexto es en el que se enmarca este trabajo cuyo objetivo fue desarrollar e implementar un sistema de visión por computador para determinar el nivel de impurezas presentes en muestras de aceite de oliva virgen en condiciones de laboratorio. Para ello, se partió del trabajo de Cano Marchal et al. (2013), una de las primeras investigaciones (sino la primera) realizadas en esta línea.

Así, los resultados de este trabajo avalan la viabilidad de emplear un sistema basado en visión artificial y reconocimiento de patrones para realizar una clasificación aproximada y rápida del contenido de las muestras de aceite de oliva. Por otra parte, se probaron cuatro vectores de características derivados del histograma de los canales de los espacios de color RGB, CIELAB y HSV, y un método de extracción de características PCA. Los clasificadores elegidos fueron las Máquinas de Soporte Vectorial, un clasificador basado en árboles de decisión (Random Forest) y la regresión logística, ya se evaluaron varios núcleos y parámetros. También se desarrolló una interfaz gráfica que permitiera al usuario cargar una imagen y obtener la predicción aproximada.

Respecto a la investigación de referencia (Cano Marchal et al., 2013) este trabajo mejoró la calidad de los datos de entrada segmentando las imágenes y probando nuevas configuraciones del vector de características. Además, se descartó el uso del método de extracción de características LDA (*Linear*

Discriminant Analysis) en previsión de que diera solo buenos resultados en el entrenamiento.

Aunque con la introducción de estos cambios en el procedimiento era de esperar un aumento de la precisión en los resultados, esto no fue así. El mejor resultado de clasificación se logró utilizando un método de extracción de características KPCA junto a un clasificador Random Forest con el vector de características derivado del histograma de los canales H, S, a y b; y se obtuvo una precisión del 65,45%, menor que el mejor resultado obtenido por Cano Marchal et al. (2013). A pesar de aplicar las mismas técnicas se han obtenido resultados muy diferentes. Esta es una cuestión que requeriría mayor investigación.

En el trabajo de Cano et al. (2013) el mejor resultado de clasificación se logró utilizando KPCA junto con una SVM de polinomio de grado 3 con un vector de entrada derivado del histograma de los canales a, b, H y S (87,66%). Sus resultados también mostraron la importancia de seleccionar un buen vector de características, ya que el uso del set 3 como vector de entrada supuso un aumento de 10 puntos en la tasa de clasificación correcta promedio para el clasificador de red neuronal artificial (ANN, *Artificial Neural Networks*) y 20 puntos para la SVM. Por otro lado, en el trabajo de Gila et al. (2020) se obtuvo una alta correlación positiva entre el método empleado actualmente para detectar impurezas y el método propuesto mediante visión artificial, con un coeficiente de determinación (R^2) de 0,996. Por último, Cano et al. (2021) consiguieron un 88% de precisión para la detección de defectos mediante regresión logística utilizando una nariz electrónica de uso general, en condiciones de laboratorio.

De forma general, los resultados obtenidos no son malos, pero están lejos de ser satisfactorios, debido especialmente a la baja precisión y la falta de robustez, que se muestra como una gran fluctuación del puntaje de clasificación

dependiendo de la semilla particular elegida para el generador de números aleatorios responsable de la partición de la validación cruzada.

No obstante, a pesar de estas limitaciones, el sistema propuesto simplifica enormemente las operaciones necesarias para realizar el análisis del contenido de impurezas en situaciones en las que no se requiere una precisión elevada. Por ejemplo, en aquellas en las que los resultados del análisis se utilizan exclusivamente para fines de supervisión de procesos. En estas situaciones, el método propuesto constituye una alternativa aproximada de bajo costo más allá de método oficial.

Finalmente, partiendo de los resultados presentados en este TFM las líneas de investigación futura deberían encaminarse en primer lugar a evaluar e identificar por qué ese bajo nivel de precisión. Además, con el fin de mejorar la precisión del sistema desarrollado, deberían emplearse estrategias ya probadas en trabajos similares. Es el caso de las estrategias introducidas en la investigación de Gila et al. (2020) en la que las muestras se centrifugaron, obteniendo dos fases separadas, el aceite en la parte superior y las impurezas no solubles en la parte inferior. O el de Cano Marchal et al. (2021) en el que emplearon un método de clasificación en dos partes, obteniendo primero las muestras pertenecientes a un clúster concreto y separando a continuación las muestras restantes.

9 Referencias bibliográficas

Acevedo, N. (2020). *Matriz de confusión en Machine Learning. Explicado paso a paso – Natalia Acevedo*. https://nataliaacevedo.com/matriz-de-confusion-en-machine-learning-explicado-paso-a-paso/

Aguilera Puerto, D., Cáceres Moreno, Ó., Martínez Gila, D. M., Gómez Ortega, J. y Gámez García, J. (2019). Online system for the identification and classification of olive fruits for the olive oil production process. *Journal of Food Measurement and Characterization, 13*(1), 716–727. https://doi.org/10.1007/s11694-018-9984-0

Aguilera Puerto, D., Martínez Gila, D. M., Gámez García, J. y Gómez Ortega, J. (2015). Sorting Olive Batches for the Milling Process Using Image Processing. *Sensors, 15*(7), 15738–15754. https://doi.org/10.3390/s150715738

Amat Rodrigo, J. (2016). *Validación de modelos predictivos (machine learning): Cross-validation, OneLeaveOut, Bootstraping*. https://www.cienciadedatos.net/documentos/30_cross-validation_oneleaveout_bootstrap

Ashwin (2020). Feature Extraction Techniques: PCA, LDA and t-SNE - Analytics Vidhya - Medium. *Analytics Vidhya*. https://medium.com/analytics-vidhya/feature-extraction-techniques-pca-lda-and-t-sne-df0459c723aa

Brownlee, J. (2021). Multinomial logistic regression with Python. *Machine Learning Mastery*. https://machinelearningmastery.com/multinomial-logistic-regression-with-python/

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167. https://doi.org/10.1023/A:1009715923555

Cano Marchal, P., Sanmartin, C., Satorres Martínez, S., Gómez Ortega, J., Mencarelli, F., & Gámez García, J. (2021). Prediction of Fruity Aroma Intensity and Defect Presence in Virgin Olive Oil Using an Electronic Nose. *Sensors*, *21*(7), 2298. https://doi.org/10.3390/s21072298

Cano Marchal, P., Martínez Gila, D., Gámez García, J. y Gómez Ortega, J. (2013). Expert system based on computer vision to estimate the content of impurities in olive oil samples. *Journal of Food Engineering*, *119*(2), 220–228. https://doi.org/10.1016/j.jfoodeng.2013.05.032

Classifier evaluation with imbalanced datasets. (2015). *Basic evaluation measures from the confusion matrix*. https://classeval.wordpress.com/introduction/basic-evaluation-measures/

Contreras, F. (2016). *Introduccion a Machine Learning*. Sunqu.

Dittakan, K., Theera-Ampornpunt, N., Witthayarat, W., Hinnoy, S., Klaiwan, S. y Pratheep, T. (2017). Banana cultivar classification using scale invariant shape analysis. En J. L. Mitrpanont y W. Sawangphol (Eds.), 2017 *2nd International Conference on Information Technology* (INCIT) (pp. 1–6). https://doi.org/10.1109/INCIT.2017.8257854

García Moreno, C. (2021). *¿Qué es el Deep Learning y para qué sirve?* | *indra*. https://www.indracompany.com/es/blogneo/deep-learning-sirve

Gatica, C. G., Best, S. S., Ceroni, J. y Lefranc, G. (2011). A New Method for Olive Fruits Recognition. En C. San Martin y S.-W. Kim (Eds.), LNCS sublibrary. SL 6, Image processing, computer vision, pattern recognition: Vol. 7042. *Progress in pattern recognition, image analysis, computer vision, and applications: 16th Iberoamerican Congress, CIARP 2011, Pucâon, Chile, November* 15-18, 2011 proceedings (Vol. 7042, pp. 646–653). Springer. https://doi.org/10.1007/978-3-642-25085-9_77

Gl4r3 (2020). Detección de colores con OpenCV y Python. *Robologs*. https://robologs.net/2020/05/22/deteccion-de-colores-con-opencv-y-python-2/

Gonzalez-Fernández, I., Iglesias-Otero, M. A., Esteki, M., Moldes, O. A., Mejuto, J. C. y Simal-Gandara, J. (2019). A critical review on the use of artificial neural networks in olive oil production, characterization and authentication. *Critical Reviews in Food Science and Nutrition, 59*(12), 1913–1926. https://doi.org/10.1080/10408398.2018.1433628

Gordillo, B., Ciaccheri, L., Mignani, A. G., Gonzalez-Miret, M. L. y Heredia, F. J. (2011). Influence of Turbidity Grade on Color and Appearance of Virgin Olive Oil. *Journal of the American Oil Chemists' Society, 88*(9), 1317–1327. https://doi.org/10.1007/s11746-011-1787-y

Gunn, S. R. (1998). Support vector machines for classification and regression. *ISIS Technical Report*, *14*.

Gupta, P. (2017). Decision Trees in Machine Learning - Towards Data Science. *Towards Data Science*. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Helwan, A., Sallam Ma'aitah, M. K., Abiyev, R. H., Uzelaltinbulat, S. y Sonyel, B. (2021). Deep Learning Based on Residual Networks for Automatic Sorting of Bananas. *Journal of Food Quality*, Article ID 5516368, 1–11. https://doi.org/10.1155/2021/5516368

Hepworth, N. J., Hammond, J. y Varley, J. (2004). Novel application of computer vision to determine bubble size distributions in beer. *Journal of Food Engineering, 61*(1), 119–124. https://doi.org/10.1016/S0260-8774(03)00193-6

Huneycutt, J. (2018). *Implementing a Random Forest Classification Model in Python*. https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

Investigación y Ciencia. (2021). *Inteligencia artificial o humana en un cerebro artificial*. https://www.investigacionyciencia.es/blogs/tecnologia/102/posts/inteligencia-artificial-o-humana-en-un-cerebro-artificial-16895

IOC. Internation Olive Council (2021). *Trade standard applying to olive oils and olive pomace oils*. (COI/T.15/NC No 3/Rev. 16). https://www.internationaloliveoil.org/wp-content/uploads/2021/07/COI-T15-NC3-REV-16-2021-_ENG.pdf

Ippolito, P. P. (2019). Feature Extraction Techniques - Towards Data Science. *Towards Data Science*. https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be

ISO. International Organization for Standardization (2017). *Animal and vegetable fats and oils. Determination of insoluble impurities content (ISO 663:2017*). https://www.iso.org/standard/71269.html

Javadikia, H., Sabzi, S. y Arribas, J. I. (2018). An automatic and non-intrusive hybrid computer vision system for the estimation of peel thickness in Thomson orange. *Spanish Journal of Agricultural Research, 16*(4). https://doi.org/10.5424/sjar/2018164-11185

Johnson, J. (2021). *Python Numpy Tutorial (with Jupyter and Colab)*. https://cs231n.github.io/python-numpy-tutorial/#numpy-arrays

Kalogianni, E. P., Georgiou, D. y Hasanov, J. H. (2019). Olive Oil Processing: Current Knowledge, Literature Gaps, and Future Perspectives*. Journal of the American Oil Chemists' Society, 96*(5), 481–507. https://doi.org/10.1002/aocs.12207

Koehrsen, W. (2018). Hyperparameter Tuning the Random Forest in Python - Towards Data Science. *Towards Data Science*. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Lenguajes de programación. (2020). *Lenguaje de programación Python. Aprende a programar*. https://lenguajesdeprogramacion.net/python/

Martínez Gila, D. M., Cano-Marchal, P., Gómez-Ortega, J. y Gámez García, J. (2018). Non-Invasive Methodology to Estimate Polyphenol Content in Extra Virgin Olive Oil Based on Stepwise Multilinear Regression. *Sensors, 18*(4). https://doi.org/10.3390/s18040975

Matplotlib development team. (2020). *Matplotlib: Python plotting — Matplotlib 3.1.2 documentation*. https://matplotlib.org/3.1.1/index.html

Moyano, M. J., Meléndez-Martínez, A. J., Alba, J. y Heredia, F. J. (2008a). A comprehensive study on the colour of virgin olive oils and its relationship with their chlorophylls and carotenoids indexes (I): CIEXYZ non-uniform colour space. *Food Research International, 41*(5), 505–512. https://doi.org/10.1016/j.foodres.2008.03.007

Moyano, M. J., Meléndez-Martínez, A. J., Alba, J. y Heredia, F. J. (2008b). A comprehensive study on the colour of virgin olive oils and its relationship with their chlorophylls and carotenoids indexes (II): CIELUV and CIELAB uniform colour spaces.

Food Research International, 41(5), 513–521. https://doi.org/10.1016/j.foodres.2008.03.006

Müller, K. R., Mika, S., Rätsch, G., Tsuda, K. y Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, *12*(2), 181–201. https://doi.org/10.1109/72.914517

Muñoz, I., Rubio-Celorio, M., Garcia-Gil, N., Guardia, M. D. y Fulladosa, E. (2015). Computer image analysis as a tool for classifying marbling: A case study in dry-cured ham. *Journal of Food Engineering, 166*, 148–155. https://doi.org/10.1016/j.jfoodeng.2015.06.004

NumPy. (2021). *NumPy*. https://numpy.org/

Olaniyi, E. O., Oyedotun, O. K., & Adnan, K. (2017). Intelligent grading system for banana fruit using neural network arbitration. *Journal of Food Process Engineering*, *40*(1), e12335. https://doi.org/10.1111/jfpe.12335

OpenCV. (2020). *OpenCV*. https://opencv.org/

Python Software Foundation. (2021). *pickle — Serialización de objetos Python - documentación de Python - 3.9.6*. https://docs.python.org/es/3/library/pickle.html

Python.org. (2021). *Python*. https://www.python.org

Rashvand, M. y Akbarnia, A. (2019). The feasibility of using image processing and artificial neural network for detecting the adulteration of sesame oil. *Aims Agriculture and Food, 4*(2), 237–243. https://doi.org/10.3934/agrfood.2019.2.237

RIP Tutorial. (2021). *Learning Scikit-learn.* https://riptutorial.com/scikit-learn

Rodríguez, D. (2019). Comparación entre la regresión logística y SVM (máquinas de vectores de soporte). *Analytics Lane*. https://www.analyticslane.com/2019/02/06/comparacion-entre-la-regresion-logistica-y-svm-maquinas-de-vectores-de-soporte/

Roy, B. (2020). All about Feature Scaling - Towards Data Science. *Towards Data Science*. https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

Sabzi, S., Abbaspour-Gilandeh, Y., & Arribas, J. I. (2017). Non-intrusive image processing Thompson orange grading methods. In *2017 56th FITCE Congress* (pp. 35-39). IEEE.

Sanaeifar, A. y Jafari, A (2019). Determination of the oxidative stability of olive oil using an integrated system based on dielectric spectroscopy and computer vision. *Information Processing in Agriculture, 6*(1), 20–25. https://doi.org/10.1016/j.inpa.2018.08.008

Sanaeifar, A., Bakhshipour, A. y La Guardia, M. de (2016). Prediction of banana quality indices from color features using support vector regression. *Talanta, 148*, 54–61. https://doi.org/10.1016/j.talanta.2015.10.073

Sanaeifar, A., Jafari, A y Golmakani, M. T. (2018). Fusion of dielectric spectroscopy and computer vision for quality characterization of olive oil during storage. *Computers and Electronics in Agriculture, 145*, 142–152. https://doi.org/10.1016/j.compag.2017.12.035

Sangwine, S. J. y Horne, R. E. N. (Eds.). (1998). *The Colour Image Processing Handbook*. Springer US; Imprint; Springer. https://doi.org/10.1007/978-1-4615-5779-1

Sawtelle, S. (2016). *Classification with SVMs and Grid Search*. https://sdsawtelle.github.io/blog/output/week7-andrew-ng-machine-learning-with-python.html

Scikit-learn developers. (2011). *User guide: contents — scikit-learn 0.11-git documentation*. https://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/user_guide.html

Scikit-learn developers. (2020a). *API Reference - scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/classes.html

Scikit-learn developers. (2020b). *Examples — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/auto_examples/index.html#

Scikit-learn developers. (2020c). *Getting Started — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/getting_started.html

Scikit-learn developers. (2020d). A tutorial on statistical-learning for scientific data processing - scikit-learn 0.24.2 documentation. https://scikit-learn.org/stable/tutorial/statistical_inference/index.html

Scikit-learn developers. (2020e). User guide: contents - scikit-learn 0.24.2 documentation. https://scikit-learn.org/stable/user_guide.html

Stone, R. (2018). Image Segmentation Using Color Spaces in OpenCV + Python. *Real Python*. https://realpython.com/python-opencv-color-spaces/

Streamlit. (2021). *Streamlit*. https://streamlit.io/

Tutor de Programación. (2017). *OpenCV Operaciones Morfológicas*. http://acodigo.blogspot.com/2017/04/opencv-operaciones-morfologicas.html

Waskom, M. (2021). *seaborn.pairplot — seaborn 0.11.1 documentation*. https://seaborn.pydata.org/generated/seaborn.pairplot.html

Zareiforoush, H., Minaei, S., Alizadeh y Banakar, A. (2015). A hybrid intelligent approach based on computer vision and fuzzy logic for quality measurement of milled rice. *Measurement, 66*, 26–34. https://doi.org/10.1016/j.measurement.2015.01.022

10 Anexos

10.1 Anexo I: Ficheros relacionados con la obtención del vector de características

10.1.1 vector.py

Funciones empleadas para la obtención del vector de características.

```
import cv2
import numpy as np


def agrupar_hist (hist,n=2):

    """Agrupa el histograma dado en función del parámetro n.

    Los parámetros de entrada de la función son:
        - hist: list. Histograma de la imagen
        - n: int. Parámetro para la agrupación del histograma. Por defecto
        toma el valor 2
    """

    hist_agrupado = []
    h =[]
    cont=0

    for pos in range(len(hist)):

        if cont<n:
            h.append(hist[pos])
            cont+=1
        else:
            cont = 0
            hist_agrupado.append(sum(h))
            h=[]
            h.append(hist[pos])
            cont+=1

    hist_agrupado.append(sum(h))
    #hist_agrupado = np.array(hist_agrupado)

    return hist_agrupado
```

```
def hist_agrupado (img,esp,canal,n=2):

    """Calcula el histograma de una imagen dada empleando la función
    agrupar_hist().

    Los parámetros de entrada de la función son:
        - img: array of unit8. Imagen de muestra en formato BGR
        - esp: string. Espacio de color en el que calcular el histograma.
        Toma los valores 'BGR', 'HSV' o 'CIELAB'. Valor por defecto esp = 'BGR'
        - canal: string. Canal en el que se calcula el histograma.
        Toma los valores 0, 1 o 2. Valor por defecto canal = 0
        - n: int. Parámetro para la agrupación del histograma.
        Por defecto toma el valor 2
    """

    hist_agrup = []

    if esp == 'BGR':
        hist = cv2.calcHist([img],[canal],None,[256],[0,256])
        hist = hist[:,0] #Esto es para pasar de un array (n,1) a una array (n,)
        hist_agrup = agrupar_hist(hist,n)

    elif esp == 'HSV':
        hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
        if canal == 0:
            hist = cv2.calcHist([hsv],[canal],None,[180],[0,180])
            hist = hist[:,0]
            hist_agrup = agrupar_hist(hist,n)

        else:
            hist = cv2.calcHist([hsv],[canal],None,[256],[0,256])
            hist = hist[:,0]
            hist_agrup = agrupar_hist(hist,n)

    elif esp == 'CIELAB':
        Lab = cv2.cvtColor(img,cv2.COLOR_BGR2Lab)
        hist = cv2.calcHist([Lab],[canal],None,[252],[0,252])
        hist = hist[:,0]
        hist_agrup = agrupar_hist(hist,n)


    return hist_agrup
```

```
def param_vector (img,esp,canal,n=2):

    """Devuelve una lista con el vector de caracterísitcas de la imagen dada
    para un espacio de color y canal especificados. Utiliza las funciones
    agrupar_hist() e hist_agrupado y sus parámetros de entrada coinciden con
    los de esta última.

    El vector de características se construye a partir de dos subconjuntos de
    parámetros: el primero se define con los valores de cada elemento del canal
    del histograma en orden descendente; el segundo se define con el índice que
    el elemento del primer subconjunto tenía en el histograma original.

    Parámetros de entrada:
        - img: imagen de muestra en formato BGR
        - esp: string. Espacio de color en el que calcular el histograma.
        Toma los valores 'BGR', 'HSV' o 'CIELAB'. Valor por defecto esp = 'BGR'
        - canal: string. Canal en el que se calcula el histograma. Toma los
        valores 0, 1 o 2. Valor por defecto canal = 0
        - n: int. Parámetro para la agrupación del histograma. Por defecto
        toma el valor 2
    """

    vector_py = []
    vpos=[]

    hist_agrup = hist_agrupado (img,esp,canal,n)

    pos =  [p for p in range(len(hist_agrup))]
    vpos = list(zip(hist_agrup,pos))

    v = sorted(vpos, reverse=True)

    elements=[]
    indexs=[]

    for i in range(len(vpos)):
        elements.append(v[i][0])
        indexs.append(v[i][1])

    vector_py = elements+indexs


    return vector_py
```

```
def img_segmentation(img,low_range,high_range,kernel_dim):
    """ Segmenta una imagen y aplica operaciones morfológicas de apertura y
    cierre al resultado.

    Parámetros de entrada:
        - img: array of unit8. Imagen de muestra en formato BGR
        - low_range: array. Valores mínimos que pueden adoptar los canales H,
        S y V respectivamente
        - high_range: array. Valores máximos que pueden adoptar los canales H,
        S y V respectivamente
        - kernel_dim: int. Dimensión del elemento estructurante para las
        operaciones morfológicas
    """

    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    kernel = np.ones((kernel_dim,kernel_dim),np.uint8)

    mask = cv2.inRange(hsv, low_range, high_range)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

    result = cv2.bitwise_and(img, img, mask=mask)
    filtered_result = [[],[],[]]

    size = np.shape(result)

    for row in range(size[0]):
        for col in range(size[1]):
            if mask[row,col] == 255:
                filtered_result[0].append([result[row,col,0]])
                filtered_result[1].append([result[row,col,1]])
                filtered_result[2].append([result[row,col,2]])

    filtered_result = np.array(filtered_result).transpose()

    return filtered_result
```

10.1.2 HistFunction.py

Funciones para la representación de histogramas.

```
# PREPARACIÓN DEL ENTORNO
import cv2
import numpy as np
from matplotlib import pyplot as plt
from vector import hist_agrupado
from vector import img_segmentation

#=====================================================
# FUNCIÓN PARA HISTOGRAMAS SIN SEGMENTACIÓN DE IMAGEN
#=====================================================

def Histogramas (n,n_H,n_Lab):

    # IMÁGENES DE MUESTRA

    path_02= "./images/i0_02__"
    path_03= "./images/i0_03__"
    path_04= "./images/i0_04__"
    path_05= "./images/i0_05__"
    path_06= "./images/i0_06__"
    path_07= "./images/i0_07__"
    path_10= "./images/i0_10__"

    images =[]
    path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
    img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo

    targets = [0,0,1,1,2,2,2] #Nivel al que pertenece cada grado de impurezas
    target_names = ['Bajo','Medio','Alto']

    # AGRUPACIÓN DE HISTOGRAMAS
    #En principio no se dividirán todos los canales entre el mismo número por
    #no tener los mismos elementos.

    labels = np.array([0,51,102,153,204,255]) #Ticks del eje x para los
canales R-G-B-S-V
    labels_H = np.array([0,30,60,90,120,150,180]) #Ticks del eje x para el
canal H
    labels_Lab = np.array([0,42,84,126,168,201,252]) #Ticks del eje x para los
canales L-a-b

    pos = labels/n
    pos_H = labels_H/n_H
    pos_Lab = labels_Lab/n_Lab

    # LECTURA DE LAS IMÁGENES Y OBTENCIÓN DE HISTOGRAMAS

    target = []
```

```
canal_B = []
canal_G = []
canal_R = []

canal_H = []
canal_S = []
canal_V = []

canal_L = []
canal_a = []
canal_b = []

col = 0 #Contador

for i in range(7):
    for img in range(img_number[i]):
        muestra = cv2.imread(path[i] + str(img) + ".jpg")
        images.append(muestra)

        #RGB
        B = hist_agrupado(muestra,'BGR',0,n)
        G = hist_agrupado(muestra,'BGR',1,n)
        R = hist_agrupado(muestra,'BGR',2,n)

        #HSV
        H = hist_agrupado(muestra,'HSV',0,n_H)
        S = hist_agrupado(muestra,'HSV',1,n)
        V = hist_agrupado(muestra,'HSV',2,n)

        #CIELAB
        L = hist_agrupado(muestra,'CIELAB',0,n_Lab)
        a = hist_agrupado(muestra,'CIELAB',1,n_Lab)
        b = hist_agrupado(muestra,'CIELAB',2,n_Lab)

        canal_B.append(B)
        canal_G.append(G)
        canal_R.append(R)

        canal_H.append(H)
        canal_S.append(S)
        canal_V.append(V)

        canal_L.append(L)
        canal_a.append(a)
        canal_b.append(b)


        target.append(targets[i])

        col+=1

canal_B = np.array(canal_B)
canal_G = np.array(canal_G)
canal_R = np.array(canal_R)
```

```
canal_H = np.array(canal_H)
canal_S = np.array(canal_S)
canal_V = np.array(canal_V)

canal_L = np.array(canal_L)
canal_a = np.array(canal_a)
canal_b = np.array(canal_b)

target = np.array(target)
target.transpose()

# REPRESENTACIÓN DE LOS CANALES H, S, a, b

colors = ['navy', 'turquoise', 'darkorange']
lw = 2
title = ['H','S', 'a', 'b']
label = [labels_H,labels,labels_Lab,labels_Lab]
x = [pos_H, pos, pos_Lab, pos_Lab]

#Agrupamos los canales a representar en un único array
canal = [canal_H, canal_S, canal_a, canal_b]

fig, axs = plt.subplots(figsize=[25, 14], nrows=3, ncols=4, sharex=False,
                        sharey=False)

for col in range(4):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)
        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(x[col])
        axs[i,col].axes.set_xticklabels(label[col])


fig.suptitle('\nCANALES H-S-a-b', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES R, G, B

title = ['B','G', 'R']

#Agrupamos los canales a representar en un único array
canal = [canal_B, canal_G, canal_R]

fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)
```

```
for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)
        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(pos)
        axs[i,col].axes.set_xticklabels(labels)

fig.suptitle('\nESPACIO DE COLOR BGR', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES H, S, V

title = ['H','S', 'V']
label = [labels_H,labels,labels]
x = [pos_H, pos, pos]

#Agrupamos los canales a representar en un único array
canal = [canal_H, canal_S, canal_V]

fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)

        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(x[col])
        axs[i,col].axes.set_xticklabels(label[col])


fig.suptitle('\nESPACIO DE COLOR HSV', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES L, a, b

title = ['L','a', 'b']

#Agrupamos los canales a representar en un único array
canal = [canal_L, canal_a, canal_b]
```

```
    fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                            sharey=False)

    for col in range(3):
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):

            hist=canal[col][target==i]
            axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
               lw=lw,label=target_name)

            axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
            target_name)

            axs[i,col].axes.set_xticks(pos_Lab)
            axs[i,col].axes.set_xticklabels(labels_Lab)

    fig.suptitle('\nESPACIO DE COLOR CIELAB', fontsize = 18)
    plt.show()

#=====================================================
# FUNCIÓN PARA HISTOGRAMAS CON SEGMENTACIÓN DE IMAGEN
#=====================================================

def HistSegmentados (n,n_H,n_Lab):

    # IMÁGENES DE MUESTRA

    path_02= "./images/i0_02__"
    path_03= "./images/i0_03__"
    path_04= "./images/i0_04__"
    path_05= "./images/i0_05__"
    path_06= "./images/i0_06__"
    path_07= "./images/i0_07__"
    path_10= "./images/i0_10__"

    images =[]
    path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
    img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo

    targets = [0,0,1,1,2,2,2] #Nivel al que pertenece cada grado de impurezas
    target_names = ['Bajo','Medio','Alto']

    # AGRUPACIÓN DE HISTOGRAMAS

    labels = np.array([0,51,102,153,204,255]) #Ticks del eje x para los
canales R-G-B-S-V
    labels_H = np.array([0,30,60,90,120,150,180]) #Ticks del eje x para el
canal H
    labels_Lab = np.array([0,42,84,126,168,201,252]) #Ticks del eje x para los
canales L-a-b

    pos = labels/n
    pos_H = labels_H/n_H
    pos_Lab = labels_Lab/n_Lab
```

```
# LECTURA DE LAS IMÁGENES Y OBTENCIÓN DE HISTOGRAMAS

target = []

canal_B = []
canal_G = []
canal_R = []

canal_H = []
canal_S = []
canal_V = []

canal_L = []
canal_a = []
canal_b = []

col = 0 #Contador

for i in range(7):
    for img in range(img_number[i]):
        image = cv2.imread(path[i] + str(img) + ".jpg")
        muestra = img_segmentation(image,(0,210,50),(40,255,255),7)
        images.append(muestra)

        #RGB
        B = hist_agrupado(muestra,'BGR',0,n)
        G = hist_agrupado(muestra,'BGR',1,n)
        R = hist_agrupado(muestra,'BGR',2,n)

        #HSV
        H = hist_agrupado(muestra,'HSV',0,n_H)
        S = hist_agrupado(muestra,'HSV',1,n)
        V = hist_agrupado(muestra,'HSV',2,n)

        #CIELAB
        L = hist_agrupado(muestra,'CIELAB',0,n_Lab)
        a = hist_agrupado(muestra,'CIELAB',1,n_Lab)
        b = hist_agrupado(muestra,'CIELAB',2,n_Lab)

        canal_B.append(B)
        canal_G.append(G)
        canal_R.append(R)

        canal_H.append(H)
        canal_S.append(S)
        canal_V.append(V)

        canal_L.append(L)
        canal_a.append(a)
        canal_b.append(b)
```

```
            target.append(targets[i])
            col+=1

    # CONVERSIÓN DE list A array PARA FACILITAR REPRESENTACIÓN GRÁFICA

    canal_B = np.array(canal_B)
    canal_G = np.array(canal_G)
    canal_R = np.array(canal_R)


    canal_H = np.array(canal_H)
    canal_S = np.array(canal_S)
    canal_V = np.array(canal_V)

    canal_L = np.array(canal_L)
    canal_a = np.array(canal_a)
    canal_b = np.array(canal_b)

    target = np.array(target)
    target.transpose()


    # REPRESENTACIÓN DE LOS CANALES H, S, a, b

    colors = ['navy', 'turquoise', 'darkorange']
    lw = 2

    title = ['H','S', 'a', 'b']
    label = [labels_H,labels,labels_Lab,labels_Lab]
    x = [pos_H, pos, pos_Lab, pos_Lab]

    #Agrupamos los canales a representar en un único array
    canal = [canal_H, canal_S, canal_a, canal_b]


    fig, axs = plt.subplots(figsize=[25, 14], nrows=3, ncols=4, sharex=False,
                            sharey=False)

    for col in range(4):
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):

            hist=canal[col][target==i]
            axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
               lw=lw,label=target_name)
            axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
            target_name)

            axs[i,col].axes.set_xticks(x[col])
            axs[i,col].axes.set_xticklabels(label[col])


    fig.suptitle('\nCANALES H-S-a-b', fontsize = 18)
    plt.show()
```

```
# REPRESENTACIÓN DE LOS CANALES R, G, B

title = ['B','G', 'R']

#Agrupamos los canales a representar en un único array
canal = [canal_B, canal_G, canal_R]

fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw)

        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(pos)
        axs[i,col].axes.set_xticklabels(labels)


fig.suptitle('\nESPACIO DE COLOR BGR', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES H, S, V

title = ['H','S', 'V']
label = [labels_H,labels,labels]
x = [pos_H, pos, pos]

#Agrupamos los canales a representar en un único array
canal = [canal_H, canal_S, canal_V]


fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)

        axs[i,col].set_title("Canal " + title[col]+ " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(x[col])
        axs[i,col].axes.set_xticklabels(label[col])
```

```
fig.suptitle('\nESPACIO DE COLOR HSV', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES L, a, b

title = ['L','a', 'b']

#Agrupamos los canales a representar en un único array
canal = [canal_L, canal_a, canal_b]

fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)

        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(pos_Lab)
        axs[i,col].axes.set_xticklabels(labels_Lab)


fig.suptitle('\nESPACIO DE COLOR CIELAB', fontsize = 18)
plt.show()

#=========================================================
# REPRESENTACIÓN AJUSTANDO EL LÍMITE DEL EJE HORIZONTAL
#=========================================================

#LÍMITES EJE HORIZONTAL

xlim_B = (0/n,40/n)
xlim_G = (50/n,150/n) #xlim_S
xlim_R = (80/n,200/n) #xlim_V

xlim_H = (10/n_H,30/n_H)

xlim_L = (50/n_Lab,180/n_Lab)
xlim_a = (120/n_Lab,150/n_Lab)
xlim_b = (160/n_Lab,200/n_Lab)

#CONFIGURACIÓN DE LAS MARCAS DEL EJE X

labels_B = np.array([0,5,10,15,20,25,30,35,40])
labels_G = np.array([60,80,100,120,140])#Ticks del eje x para canales G, S
```

```
    labels_R = np.array([80,100,120,140,160,180,200]) #Ticks del eje x para
los canales R y V

    labels_H = np.array([10,12,14,16,18,20,22,24,26,28,30])

    labels_L = np.array([60,80,100,120,140,160,180])
    labels_a = np.array([120,125,130,135,140,145,150])
    labels_b = np.array([160,165,170,175,180,185,190,195,200])

    pos_B = labels_B/n
    pos_G = labels_G/n
    pos_R = labels_R/n

    pos_H = labels_H/n_H

    pos_L = labels_L/n_Lab
    pos_a = labels_a/n_Lab
    pos_b = labels_b/n_Lab

    # REPRESENTACIÓN DE LOS CANALES H, S, a, b
    title = ['H','S', 'a', 'b']
    xlim = (xlim_H, xlim_G, xlim_a, xlim_b)
    label = [labels_H,labels_G,labels_a,labels_b]
    x = [pos_H, pos_G, pos_a, pos_b]

    #Agrupamos los canales a representar en un único array
    canal = [canal_H, canal_S, canal_a, canal_b]


    fig, axs = plt.subplots(figsize=[25, 14], nrows=3, ncols=4, sharex=False,
                            sharey=False)

    for col in range(4):
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):

            hist=canal[col][target==i]
            axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
               lw=lw,label=target_name)
            axs[i,col].set_title("Canal " + title[col]+ " - Nivel " +
            target_name)

            axs[i,col].axes.set_xticks(x[col])
            axs[i,col].axes.set_xticklabels(label[col])

            axs[i,col].axes.set_xlim(xlim[col][0], xlim[col][1])


    fig.suptitle('\nCANALES H-S-a-b', fontsize = 18)
    plt.show()
```

```
# REPRESENTACIÓN DE LOS CANALES R, G, B

title = ['B','G', 'R']
xlim = (xlim_B, xlim_G, xlim_R)
label = [labels_B,labels_G,labels_R]
x = [pos_B, pos_G, pos_R]

#Agrupamos los canales a representar en un único array
canal = [canal_B, canal_G, canal_R]

fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw)

        axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
        target_name)

        axs[i,col].axes.set_xticks(x[col])
        axs[i,col].axes.set_xticklabels(label[col])

        axs[i,col].axes.set_xlim(xlim[col][0], xlim[col][1])


fig.suptitle('\nESPACIO DE COLOR BGR', fontsize = 18)
plt.show()


# REPRESENTACIÓN DE LOS CANALES H, S, V

title = ['H','S', 'V']
xlim = (xlim_H, xlim_G, xlim_R)
label = [labels_H,labels_G,labels_R]
x = [pos_H, pos_G, pos_R]

#Agrupamos los canales a representar en un único array
canal = [canal_H, canal_S, canal_V]


fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                        sharey=False)

for col in range(3):
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):

        hist=canal[col][target==i]
        axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
           lw=lw,label=target_name)
```

```
            axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
            target_name)

            axs[i,col].axes.set_xticks(x[col])
            axs[i,col].axes.set_xticklabels(label[col])

            axs[i,col].axes.set_xlim(xlim[col][0], xlim[col][1])


    fig.suptitle('\nESPACIO DE COLOR HSV', fontsize = 18)
    plt.show()


    # REPRESENTACIÓN DE LOS CANALES L, a, b

    title = ['L','a', 'b']
    xlim = (xlim_L, xlim_a, xlim_b)
    label = [labels_L,labels_a,labels_b]
    x = [pos_L, pos_a, pos_b]

    #Agrupamos los canales a representar en un único array
    canal = [canal_L, canal_a, canal_b]

    fig, axs = plt.subplots(figsize=[20, 14], nrows=3, ncols=3, sharex=False,
                            sharey=False)

    for col in range(3):
        for color, i, target_name in zip(colors, [0, 1, 2], target_names):

            hist=canal[col][target==i]
            axs[i,col].plot(hist.transpose(), color=color, alpha=.8,
               lw=lw,label=target_name)

            axs[i,col].set_title("Canal " + title[col] + " - Nivel " +
            target_name)

            axs[i,col].axes.set_xticks(x[col])
            axs[i,col].axes.set_xticklabels(label[col])

            axs[i,col].axes.set_xlim(xlim[col][0], xlim[col][1])


    fig.suptitle('\nESPACIO DE COLOR CIELAB', fontsize = 18)
    plt.show()
```

10.1.3 PairPlot_PCA.py

Visualización de la separación entre clases.

Este fichero emplea el módulo el módulo PairPlot en las cuatro configuraciones del vector de características para visualizar la separación entre las clases tras aplicar el método de extracción de características PCA.

```
import pickle
import seaborn as sns
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA


#================================
# VECTOR DE CARACTERÍSTICAS GRLab
#================================

filename = 'DataBase'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

level=[]

for i in range(len(y)):
    level.append(target_names[y[i]])

scaler = MinMaxScaler()
X_s = scaler.fit_transform(X)

pca = PCA(n_components = 6)
X_r = pca.fit_transform(X_s)


d={'level':level, 'PCA 1': X_r[:,0], 'PCA 2': X_r[:,1], 'PCA 3': X_r[:,2],
   'PCA 4': X_r[:,3],'PCA 5': X_r[:,4], 'PCA 6': X_r[:,5]}
df = pd.DataFrame(data=d)

fig = sns.pairplot(df, hue='level')
fig.savefig('Results/PairPlot_PCA.png')
```

```
#================================
# VECTOR DE CARACTERÍSTICAS HSab
#================================

filename = 'DataBase_HSab'
infile = open(filename,'rb')

DataBase_HSab = pickle.load(infile)
infile.close()

X = DataBase_HSab.data
y = DataBase_HSab.target
target_names = DataBase_HSab.target_names

level=[]

for i in range(len(y)):
    level.append(target_names[y[i]])


X_s = scaler.fit_transform(X)

X_r = pca.fit_transform(X_s)


d={'level':level, 'PCA 1': X_r[:,0], 'PCA 2': X_r[:,1], 'PCA 3': X_r[:,2],
   'PCA 4': X_r[:,3],'PCA 5': X_r[:,4], 'PCA 6': X_r[:,5]}
df = pd.DataFrame(data=d)

fig = sns.pairplot(df, hue='level')
fig.savefig('Results/PairPlot_PCA_HSab.png')

#========================================
# VECTOR DE CARACTERÍSTICAS HSab REORDENADO
#========================================
filename = 'DataBase_HSab_Reordenado'
infile = open(filename,'rb')

DataBase_inicial = pickle.load(infile)
infile.close()

X = DataBase_inicial.data
y = DataBase_inicial.target
target_names = DataBase_inicial.target_names

#level = np.zeros(len(y))
level=[]

for i in range(len(y)):
    level.append(target_names[y[i]])
```

```
X_s = scaler.fit_transform(X)

X_r = pca.fit_transform(X_s)


d={'level':level, 'PCA 1': X_r[:,0], 'PCA 2': X_r[:,1], 'PCA 3': X_r[:,2],
   'PCA 4': X_r[:,3],'PCA 5': X_r[:,4], 'PCA 6': X_r[:,5]}
df = pd.DataFrame(data=d)

fig = sns.pairplot(df, hue='level')
fig.savefig('Results/PairPlot_PCA_Reordenado.png')

#================================
# VECTOR DE CARACTERÍSTICAS GRL
#================================

filename = 'DataBase_GRL'
infile = open(filename,'rb')

DataBase_GRL = pickle.load(infile)
infile.close()

X = DataBase_GRL.data
y = DataBase_GRL.target
target_names = DataBase_GRL.target_names

level=[]

for i in range(len(y)):
    level.append(target_names[y[i]])


X_s = scaler.fit_transform(X)

X_r = pca.fit_transform(X_s)


d={'level':level, 'PCA 1': X_r[:,0], 'PCA 2': X_r[:,1], 'PCA 3': X_r[:,2],
   'PCA 4': X_r[:,3],'PCA 5': X_r[:,4], 'PCA 6': X_r[:,5]}
df = pd.DataFrame(data=d)

fig = sns.pairplot(df, hue='level')
fig.savefig('Results/PairPlot_PCA_GRL.png')
```

10.1.4 DataBase_HSab_Inicial.py

Obtención del vector de características HSab Inicial.

En este fichero se extraen las características de las imágenes de muestra. El vector de características empleado encadena el vector correspondiente a los canales H, S, a y b, en dos subconjuntos.

Una vez obtenida la matriz de características, se almacenan los datos en una estructura de datos similar al diccionario (bunch) que facilita el manejo de conjuntos de datos. Los elementos de este objeto son:

- **data** (*array*): matriz de características
- **features** (*list*): nombre de los canales empleados
- **level_names** (list): nombre de los distintos porcentajes de impurezas insolubles
- **target_level** (*array*): nivel de impurezas correspondiente a cada muestra
- **target** (*array*): grado de impurezas que corresponde a cada muestra (%)
- **target_names** (*list*): nombres del grado de impurezas (bajo, medio o alto)

```
import cv2
import pickle
import numpy as np
from vector import param_vector
from vector import img_segmentation
from sklearn.utils import Bunch


path_02= "./images/i0_02__"
path_03= "./images/i0_03__"
path_04= "./images/i0_04__"
path_05= "./images/i0_05__"
path_06= "./images/i0_06__"
path_07= "./images/i0_07__"
path_10= "./images/i0_10__"

images =[]
path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo
targets = [0,0,1,1,2,2,2]
```

```
p_vector=[]
target=[]
target_level=[]


for i in range(7):
    for img in range(img_number[i]):
        image = cv2.imread(path[i] + str(img) + ".jpg")
        print(img)
        muestra = img_segmentation(image,(0,210,50),(40,255,255),7)
        images.append(muestra)

        #VECTOR DE CARACTERÍSTICAS

        #Histogramas que componen el vector de características:
        H = param_vector(muestra,'HSV',0,4)
        S = param_vector(muestra,'HSV',1,4)
        a = param_vector(muestra,'CIELAB',1,4)
        b = param_vector(muestra,'CIELAB',2,4)

        vector = H+S+a+b #Se juntan los histogramas reordenados en un
solo vector
        vector = np.array(vector, int)

        p_vector.append(vector) #Matriz de características

        target.append(targets[i]) #Etiquetas de las muestras
        target_level.append(i)


target=np.array(target)
p_vector = np.array(p_vector)
target_names = ['Bajo','Medio','Alto']
levels_names = ['0.2','0.3','0.4','0.5','0.6','0.7','1.0']

features_names = ['H','S','a','b']
features = []

f_number = [60,64,56,56] #Nº de elementos que contiene cada canal
f = 0


for i in range(4):
    for j in range(f_number[i]):
        features.append(f)
    f+=1

feat_array = np.array(features)
```

```
#ESTRUCTURA DE DATOS:
DataBase = Bunch(data=p_vector, target=target,
target_names=target_names,
          taget_level=target_level,levels_names=levels_names,
          features=features, features_names=features_names)

filename = 'DataBase_HSab_Inicial'
outfile = open(filename,'wb')

pickle.dump(DataBase,outfile)
outfile.close()
```

10.1.5 DataBase_HSab.py

Obtención del vector de características HSab.

En este fichero se extraen las características de las imágenes de muestra. El vector de características empleado encadena el vector correspondiente a los canales H, S, a y b, en ese orden.

Una vez obtenida la matriz de características, se almacenan los datos en una estructura de datos similar al diccionario (bunch) que facilita el manejo de conjuntos de datos. Los elementos de este objeto son:

- **data** (*array*): matriz de características
- **features** (*list*): nombre de los canales empleados
- **level_names** (list): nombre de los distintos porcentajes de impurezas insolubles
- **target_level** (*array*): nivel de impurezas correspondiente a cada muestra
- **target** (*array*): grado de impurezas que corresponde a cada muestra (%)
- **target_names** (*list*): nombres del grado de impurezas (bajo, medio o alto)

```
import cv2
import pickle
import numpy as np
from vector import hist_agrupado
from vector import img_segmentation
from sklearn.utils import Bunch

#IMÁGENES DE MUESTRA

path_02= "./images/i0_02__"
path_03= "./images/i0_03__"
path_04= "./images/i0_04__"
path_05= "./images/i0_05__"
path_06= "./images/i0_06__"
path_07= "./images/i0_07__"
path_10= "./images/i0_10__"

images =[]
path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo
targets = [0,0,1,1,2,2,2]

p_vector=[]
target=[]
target_level=[]

n = 4
n_Lab = 4

for i in range(7):
    for img in range(img_number[i]):
        image = cv2.imread(path[i] + str(img) + ".jpg")
        print(img)
        muestra = img_segmentation(image,(0,210,50),(40,255,255),7)
        images.append(muestra)

        #VECTOR DE CARACTERÍSTICAS

        #Histogramas que componen el vector de características:
        H = hist_agrupado(muestra,'HSV',0,n)
        S = hist_agrupado(muestra,'HSV',1,n)
        a = hist_agrupado(muestra,'CIELAB',1,n_Lab)
        b = hist_agrupado(muestra,'CIELAB',2,n_Lab)

        vector = H+S+a+b #Se juntan los histogramas en un solo
vector
        vector = np.array(vector, int)

        p_vector.append(vector) #Matriz de características

        target.append(targets[i]) #Etiquetas de las muestras
        target_level.append(i)
```

```
target=np.array(target)
p_vector = np.array(p_vector)

target_names = ['Bajo','Medio','Alto']
levels_names = ['0.2','0.3','0.4','0.5','0.6','0.7','1.0']

features_names = ['H','S','a','b']
features = []

f_number = [45,64,63,63] #Nº de elementos que contiene cada canal
f = 0


for i in range(4):
    for j in range(f_number[i]):
        features.append(f)
    f+=1

feat_array = np.array(features)


#ESTRUCTURA DE DATOS:
DataBase = Bunch(data=p_vector, target=target,
target_names=target_names,
          taget_level=target_level,levels_names=levels_names,
          features=features, features_names=features_names)

filename = 'DataBase_HSab'
outfile = open(filename,'wb')

pickle.dump(DataBase,outfile)
outfile.close()
```

10.1.6 DataBase_GRLab.py

Obtención del vector de características GRLab.

En este fichero se extraen las características de las imágenes de muestra. El vector de características empleado encadena el histograma agrupado correspondiente a los canales G, R, L, a y b, en ese orden.

Una vez obtenida la matriz de características, se almacenan los datos en una estructura de datos similar al diccionario (bunch) que facilita el manejo de conjuntos de datos. Los elementos de este objeto son:

- **data** (*array*): matriz de características
- **features** (*list*): nombre de los canales empleados
- **level_names** (list): nombre de los distintos porcentajes de impurezas insolubles
- **target_level** (*array*): nivel de impurezas correspondiente a cada muestra
- **target** (*array*): grado de impurezas que corresponde a cada muestra (%)
- **target_names** (*list*): nombres del grado de impurezas (bajo, medio o alto)

```
import cv2
import pickle
import numpy as np
from vector import hist_agrupado
from vector import img_segmentation
from sklearn.utils import Bunch

#IMÁGENES DE MUESTRA

path_02= "./images/i0_02__"
path_03= "./images/i0_03__"
path_04= "./images/i0_04__"
path_05= "./images/i0_05__"
path_06= "./images/i0_06__"
path_07= "./images/i0_07__"
path_10= "./images/i0_10__"

images =[]
path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo
targets = [0,0,1,1,2,2,2]

p_vector=[]
target=[]
target_level=[]

n = 4
n_Lab = 4

for i in range(7):
    for img in range(img_number[i]):
        image = cv2.imread(path[i] + str(img) + ".jpg")
        print(img)
        muestra = img_segmentation(image,(0,210,50),(40,255,255),7)
        images.append(muestra)
```

```
        #VECTOR DE CARACTERÍSTICAS

        #Histogramas que componen el vector de características:
        G = hist_agrupado(muestra,'BGR',1,n)
        R = hist_agrupado(muestra,'BGR',2,n)
        L = hist_agrupado(muestra,'CIELAB',0,n_Lab)
        a = hist_agrupado(muestra,'CIELAB',1,n_Lab)
        b = hist_agrupado(muestra,'CIELAB',2,n_Lab)

        vector = G+R+L+a+b #Se juntan los histogramas en un solo
vector
        vector = np.array(vector, int)

        p_vector.append(vector)  #Matriz de características

        target.append(targets[i]) #Etiquetas de las muestras
        target_level.append(i)


target=np.array(target)
p_vector = np.array(p_vector)


target_names = ['Bajo','Medio','Alto']
levels_names = ['0.2','0.3','0.4','0.5','0.6','0.7','1.0']

features_names = ['G','R','L','a','b']
features = []

f_number = [64,64,63,63,63] #Nº de elementos de cada canal
f = 0


for i in range(5):
    for j in range(f_number[i]):
        features.append(f)
    f+=1

feat_array = np.array(features)


#ESTRUCTURA DE DATOS:
DataBase = Bunch(data=p_vector, target=target,
target_names=target_names,
          taget_level=target_level,levels_names=levels_names,
          features=features, features_names=features_names)

filename = 'DataBase_GRLab'
outfile = open(filename,'wb')

pickle.dump(DataBase,outfile)
outfile.close()
```

10.1.7 DataBase_GRL.py

Obtención del vector de características GRL.

En este fichero se extraen las características de las imágenes de muestra. El vector de características empleado encadena el vector correspondiente a los canales G, R y L en ese orden.

Una vez obtenida la matriz de características, se almacenan los datos en una estructura de datos similar al diccionario (bunch) que facilita el manejo de conjuntos de datos. Los elementos de este objeto son:

- **data** (*array*): matriz de características
- **features** (*list*): nombre de los canales empleados
- **level_names** (list): nombre de los distintos porcentajes de impurezas insolubles
- **target_level** (*array*): nivel de impurezas correspondiente a cada muestra
- **target** (*array*): grado de impurezas que corresponde a cada muestra (%)
- **target_names** (*list*): nombres del grado de impurezas (bajo, medio o alto)

```
import cv2
import pickle
import numpy as np
from vector import hist_agrupado
from vector import img_segmentation
from sklearn.utils import Bunch

#IMÁGENES DE MUESTRA

path_02= "./images/i0_02__"
path_03= "./images/i0_03__"
path_04= "./images/i0_04__"
path_05= "./images/i0_05__"
path_06= "./images/i0_06__"
path_07= "./images/i0_07__"
path_10= "./images/i0_10__"

images =[]
path = [path_02,path_03,path_04,path_05,path_06,path_07,path_10]
img_number =  [8,79,41,13,16,6,2] #Nº de imágenes de cada tipo
targets = [0,0,1,1,2,2,2]
```

```
p_vector=[]
target=[]
target_level=[]

n = 4
n_Lab = 4

for i in range(7):
    for img in range(img_number[i]):
        image = cv2.imread(path[i] + str(img) + ".jpg")
        print(img)
        muestra = img_segmentation(image,(0,210,50),(40,255,255),7)
        images.append(muestra)

        #VECTOR DE CARACTERÍSTICAS

        #Histogramas que componen el vector de características:
        G = hist_agrupado(muestra,'BGR',1,n)
        R = hist_agrupado(muestra,'BGR',2,n)
        L = hist_agrupado(muestra,'CIELAB',0,n_Lab)

        vector = G+R+L #Se juntan los histogramas en un solo vector
        vector = np.array(vector, int)

        p_vector.append(vector) #Matriz de características

        target.append(targets[i]) #Etiquetas de las muestras
        target_level.append(i)


target=np.array(target)
p_vector = np.array(p_vector)


target_names = ['Bajo','Medio','Alto']
levels_names = ['0.2','0.3','0.4','0.5','0.6','0.7','1.0']

features_names = ['G','R','L']
features = []

f_number = [64,64,63] #Nº de elementos que contiene cada canal
f = 0


for i in range(3):
    for j in range(f_number[i]):
        features.append(f)
    f+=1

feat_array = np.array(features)
```

```
#ESTRUCTURA DE DATOS:
DataBase = Bunch(data=p_vector, target=target,
target_names=target_names,
          taget_level=target_level,levels_names=levels_names,
          features=features, features_names=features_names)

filename = 'DataBase_GRL'
outfile = open(filename,'wb')

pickle.dump(DataBase,outfile)
outfile.close()
```

10.2 Anexo II: Ficheros relacionados con el ajuste de hiperparámetros

10.2.1 LinearSVC_nestedCV.py

Ajuste de los hiperparámetros de una Máquina de Soporte Vectorial Lineal mediante validación cruzada anidada. El valor óptimo se obtiene mediante búsqueda de cuadrícula (Grid-search).

```
#CONFIGURACIÓN DEL ENTORNO

import pickle
import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

from sklearn.svm import LinearSVC

from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab_Inicial' #Modificar para cada vector de
características
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#ESTIMADOR Y PARÁMETRO

svm = LinearSVC(random_state = 42)

#Valores que adoptará el hiperparámetro C
param_C = np.logspace(-2, 3, 6)

param_grid = dict (C = param_C)

#Validación cruzada k-Fold
kf = KFold(n_splits=5, shuffle=True, random_state = 42)

grid = GridSearchCV(svm, param_grid=param_grid,scoring='accuracy', cv=kf,
                    return_train_score=True)
```

```
#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values = [] #Vector en el que se almacena el valor óptimo de la iteración
test_scores = [] #Vector en el que se almacena la precisión de la validación
de cada iteración

#MÉTODOS DE ESCALADO Y EXTRACCIÓN DE CARACTERÍSTICAS
scaler = MinMaxScaler()
pca = PCA(n_components=6)
PCA_components = [] #Aquí se almacenan los valores de las componentes del PCA


for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS
    X_r = pca.fit_transform(X_s)
    PCA_components.append(pca.components_)

    #GRID SEARCH

    grid.fit(X_r,y_train)

    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']

    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']

    params = grid.cv_results_['params']


    plt.semilogx(param_C, tr_mean, linestyle="-", color="darkorange",
                 label="training score")

    plt.semilogx(param_C, cv_mean, linestyle="-", color="b",
                 label="cross-validation score")

    lw = 2

    plt.fill_between(param_C, tr_mean - tr_std, tr_mean + tr_std, alpha=0.2,
                 color="darkorange", lw=lw)
    plt.fill_between(param_C, cv_mean - cv_std, cv_mean + cv_std, alpha=0.2,
                 color="b", lw=lw)
```

```
    title = 'Grid Search. Iteración ' + str(it)
    plt.title(title)
    plt.xlabel("C")
    plt.ylabel("Accuracy")
    plt.ylim(0.0, 1.1)
    plt.legend(loc="best")

    plt.savefig('Results/LinearSVC_GS_it'+ str(it) + '.png')
    plt.show()


    value = grid.best_params_['C']
    values.append(value)
    print('\n' + 'El valor óptimo de C obtenido con Grid Search es: ', value)
    print('Mejor precisión alcanzada: {0:0.4f}, con C = {1}'
          .format(grid.best_score_, grid.best_params_['C']))

    #VALIDACIÓN
    X_s = scaler.fit_transform(X_test)
    X_test = pca.fit_transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'.
           format(test_score) + '\n')


    it+=1


test_mean = np.mean(test_scores)
test_std = np.std(test_scores)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std))
```

10.2.2 SVC_rbf_nestedCV

El valor óptimo de los hiperparámetros se obtiene mediante búsqueda de cuadrícula (Grid-search).

En este programa se aplica una validación cruzada anidada. Se ejecutan dos ciclos de validación cruzada en paralelo: uno por el estimador GridSearchCV para fijar los hiperparámetros C y gamma, y otro por el método cross_val_score para medir la exactitud de la predicción del estimador. Las puntuaciones

resultantes son estimaciones no sesgadas de la puntuación de predicción sobre nuevos datos.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle
import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab_Inicial'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#ESTIMADOR Y PARÁMETRO

svm = SVC(random_state=42)

#Valores que adoptará el hiperparámetro gamma
param_gamma = np.logspace(-2, 2, 5)

param_grid = dict (gamma = param_gamma)

#Validación cruzada k-Fold
kf = KFold(n_splits=5, shuffle=True, random_state = 42)

grid = GridSearchCV(svm, param_grid=param_grid,scoring='accuracy', cv=kf,
                    return_train_score=True)


#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values = [] #Vector en el que se almacena el valor óptimo de la iteración
test_scores = [] #Vector en el que se almacena la precisión del conjunto
de validación de cada iteración
```

```
#MÉTODOS DE ESCALADO Y EXTRACCIÓN DE CARACTERÍSTICAS
scaler = MinMaxScaler()
pca = PCA(n_components=6)

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS
    X_r = pca.fit_transform(X_s)

    #GRID SEARCH

    grid.fit(X_r,y_train)


    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']

    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']


    plt.semilogx(param_gamma, tr_mean, linestyle="-", color="darkorange",
                 label="training score")
    plt.semilogx(param_gamma, cv_mean, linestyle="-", color="b",
                 label="cross-validation score")

    lw = 2

    plt.fill_between(param_gamma, tr_mean - tr_std, tr_mean + tr_std,
                     alpha=0.2, color="darkorange", lw=lw)
    plt.fill_between(param_gamma, cv_mean - cv_std, cv_mean + cv_std,
                     alpha=0.2, color="b", lw=lw)

    title = 'Grid Search. Iteración ' + str(it)
    plt.title(title)
    plt.xlabel(r"$\gamma$")
    plt.ylabel("Score")
    plt.legend(loc="best")

    plt.savefig('Results/SVC_rbf_GS_it'+ str(it) + '.png')
    plt.show()

    value = grid.best_params_['gamma']
    values.append(value)
    print('\n' + 'El valor óptimo de gamma con Grid Search es: ', value)
    print('Mejor precisión alcanzada: {0:0.4f}, con gamma = {1}'
          .format(grid.best_score_, grid.best_params_['gamma']))
```

```
    #COMPROBACIÓN
    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'
           .format(test_score) + '\n')


    it+=1

test_mean = np.mean(test_scores)
test_std = np.std(test_scores)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std))

#=======================================
# AJUSTE DE LOS HIPERPARÁMTEROS C Y GAMMA
#=======================================

#Modificación de la cuadrícula de parámetros para ajustar los dos
hiperparámetros

param_C = np.logspace(-2, 3, 6)

param_grid = dict (C = param_C, gamma = param_gamma)

grid = GridSearchCV(svm, param_grid=param_grid,scoring='accuracy', cv=kf,
                    return_train_score=True)

#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values_gamma= [] #Vector en el que se almacena el valor óptimo de gamma
de la iteración
values_C= [] #Vector en el que se almacena el valor óptimo de C de la
iteración
test_scores_= [] #Vector en el que se almacena la score del conjunto de
validación de cada iteración


for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    X_r = pca.fit_transform(X_s)

    #GRID SEARCH

    grid.fit(X_r,y_train)


    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']

    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']

    params = grid.cv_results_['params']

    value_C = grid.best_params_['C']
    value_gamma = grid.best_params_['gamma']
    values_gamma.append(value_gamma)
    values_C.append(value_C)

    print('\n' + 'ITERACIÓN {0}'.format(it))
    print('El valor óptimo de gamma con Grid Search es: ', value_gamma)
    print('El valor óptimo de C con Grid Search es: ', value_C)
    print('Mejor precisión: {0:0.4f}, con gamma = {1} y C = {2}'
          .format(grid.best_score_, grid.best_params_['gamma'],
                  grid.best_params_['C']))

    #COMPROBACIÓN
    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores_.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'
           .format(test_score) + '\n')


    it+=1

test_mean_ = np.mean(test_scores_)
test_std_ = np.std(test_scores_)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean_))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std_))
```

10.2.3 SVC_poly_nestedCV

El valor óptimo se obtiene mediante búsqueda de cuadrícula (Grid-search).

En este programa se aplica una validación cruzada anidada. Se ejecutan dos ciclos de validación cruzada en paralelo: uno por el estimador GridSearchCV para fijar los hiperparámetros C y degree, y otro por el método cross_val_score con el que se mide la exactitud de la predicción del estimador. Las puntuaciones resultantes son estimaciones no sesgadas de la puntuación de predicción sobre nuevos datos.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle
import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

filename = 'DataBase_GRL' #Modificar para cada vector de características
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#ESTIMADOR Y PARÁMETRO

svm = SVC(kernel = 'poly', random_state = 42)

#Valores que adoptará el metaparámetro degree
param_degree = [2,3,4,5,6]

param_grid = dict (degree = param_degree)
```

```
#Validación cruzada k-Fold
kf = KFold(n_splits=5, shuffle=True, random_state = 42)

grid = GridSearchCV(svm, param_grid = param_grid, scoring ='accuracy',
                    cv=kf, return_train_score=True)


#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values = [] #Vector en el que se almacena el valor óptimo de la iteración
test_scores = [] #Vector en el que se almacena la score del conjunto de
validación de cada iteración

#MÉTODOS DE ESCALADO Y EXTRACCIÓN DE CARACTERÍSTICAS
scaler = MinMaxScaler()
pca = PCA(n_components=6)

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS
    X_r = pca.fit_transform(X_s)

    #GRID SEARCH

    grid.fit(X_r,y_train)


    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']

    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']

    params = grid.cv_results_['params']


    plt.plot(param_degree, tr_mean, linestyle="-", color="darkorange",
             label="training score")
    plt.plot(param_degree, cv_mean, linestyle="-", color="b",
             label="cross-validation score")

    lw = 2
```

```
    plt.fill_between(param_degree, tr_mean - tr_std, tr_mean + tr_std,
                     alpha=0.2, color="darkorange", lw=lw)
    plt.fill_between(param_degree, cv_mean - cv_std, cv_mean + cv_std,
                     alpha=0.2, color="b", lw=lw)

    title = 'Grid Search. Iteración ' + str(it)
    plt.title(title)
    plt.xlabel("Degree")
    plt.ylabel("Score")
    plt.legend(loc="best")

    plt.savefig('Results/SVC_poly_GS_it'+ str(it) + '.png')
    plt.show()

    value = grid.best_params_['degree']
    values.append(value)
    print('\n'+'El valor óptimo del grado obtenido con Grid Search es: ', value)
    print('Mejor precisión alcanzada: {0:0.4f}, con grado = {1}'
          .format(grid.best_score_, grid.best_params_['degree']))

    #COMPROBACIÓN
    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'
           .format(test_score) + '\n')


    it+=1

test_mean = np.mean(test_scores)
test_std = np.std(test_scores)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std))

#=======================================
# AJUSTE DE LOS HIPERPARÁMTEROS C Y DEGREE
#=======================================

#Modificación de la cuadrícula de parámetros para ajustar los dos
hiperparámetros

param_C = np.logspace(-2, 3, 6)

param_grid = dict (C = param_C, degree = param_degree)

grid = GridSearchCV(svm, param_grid=param_grid,scoring='accuracy', cv=kf,
                    return_train_score=True)
```

```
#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values_degree= [] #Vector en el que se almacena el valor óptimo del grado de la
iteración
values_C= [] #Vector en el que se almacena el valor óptimo de C de la iteración
test_scores_= [] #Vector en el que se almacena la score del conjunto de
validación de cada iteración

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    X_r = pca.fit_transform(X_s)

    #GRID SEARCH

    grid.fit(X_r,y_train)


    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']

    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']

    params = grid.cv_results_['params']


    value_C = grid.best_params_['C']
    value_degree = grid.best_params_['degree']
    values_degree.append(value_degree)
    values_C.append(value_C)

    print('\n' + 'ITERACIÓN {0}'.format(it))
    print('El valor óptimo del grado con Gid Search es: ', value_degree)
    print('El valor óptimo de C con Gid Search es: ', value_C)
    print('Mejor precisión: {0:0.4f}, con grado {1} y C = {2}'
          .format(grid.best_score_, grid.best_params_['degree'],
                  grid.best_params_['C']))
```

```
    #COMPROBACIÓN
    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores_.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'
           .format(test_score) + '\n')


    it+=1

test_mean_ = np.mean(test_scores_)
test_std_ = np.std(test_scores_)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean_))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std_))
```

10.2.4 RandomForest_PatameterGrid.py

Ajuste inicial de los hiperparámetros del clasificador Random Forest aplicando una búsqueda aleatoria con validación cruzada (RandomizedSearchCV).

```
#CONFIGURACIÓN DEL ENTORNO

import pickle
import numpy as np

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab_Inicial' #Modificar para cada vector de
características
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()
```

```
X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

pca = PCA(n_components=6)
scaler = MinMaxScaler()

#ESTIMADOR

rf = RandomForestClassifier(random_state = 42)

#CUADRÍCULA DE PARÁMETROS

# Número de árboles en random forest
n_estimators = [int(x) for x in np.linspace(start = 200,stop = 2000,num = 10)]
# Número de características a tener en cuenta en cada división
max_features = ['auto', 'sqrt']
# Número máximo de niveles en los árboles
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Número mínimo de muestras requeridas en cada nodo
min_samples_split = [2, 5, 10]
# Número mínimo de muestras requeridas en cada nodo hoja
min_samples_leaf = [1, 2, 4]
# Método de selección de muestras para entrenar cada árbol
bootstrap = [True, False]

param_grid = dict (n_estimators = n_estimators, max_features = max_features,
               max_depth = max_depth, min_samples_split = min_samples_split,
               min_samples_leaf = min_samples_leaf, bootstrap = bootstrap)

#SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)
kf_3 = KFold(n_splits = 3, shuffle = True, random_state = 42)


random_search = RandomizedSearchCV (estimator = rf,
                                    param_distributions = param_grid,
                                    n_iter = 100, cv = kf_3,
                                    random_state = 42)

#CLASIFICADOR ÓPTIMO
it = 1

for train_index, test_index in kf.split(X):

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    X_s = scaler.fit_transform(X_train)
```

```
    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    X_r = pca.fit_transform(X_s)


    #RANDOM SEARCH CV

    random_search.fit(X_r, y_train)
    best_model = random_search.best_estimator_

    print('\n' + 'ITERACIÓN {0}'.format(it))
    print('Mejor precisión alcanzada: {0:0.4f}, con parámetros = {1}'
          .format(random_search.best_score_, random_search.best_params_)+'\n')

    # COMPARACIÓN DEL MEJOR CLASIFICADOR OBTENIDO CON CLASIFICADOR BASE

    rf_base = RandomForestClassifier()
    rf_base.fit(X_r, y_train) # PARÁMETROS POR DEFECTO

    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    base_score = rf_base.score(X_test, y_test)
    best_score = random_search.score(X_test, y_test)

    print ('Precisión de validación del modelo base: {0:0.4f}'
           .format(base_score))
    print ('Precisión de validación del mejor modelo obtenido:{0:0.4f}'
           .format(best_score))

    it+=1
```

10.2.5 RandomForest_HSab.py

Ajuste de los hiperparámetros del clasificador Random Forest aplicando una búsqueda de rejilla con validación cruzada (GridSearchCV) con el vector de características HSab.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
```

```
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#ESCALADO DEL VECTOR DE CARACTERÍSTICAS

scaler = MinMaxScaler()
X_s = scaler.fit_transform(X_train)

#MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

pca = PCA(n_components=6)
X_r = pca.fit_transform(X_s)

#ESTIMADOR

rf = RandomForestClassifier(random_state = 42)

#CUADRÍCULA DE PARÁMETROS

# Número de árboles en random forest
n_estimators = [1400, 1500, 1600, 2000]
# Número de características a tener en cuenta en cada división
max_features = ['auto', 'sqrt']
# Número máximo de niveles en los árboles
max_depth = [10, 50, 80, 90, 100]
max_depth.append(None)
# Número mínimo de muestras requeridas en cada nodo
min_samples_split = [2, 5, 10]
# Número mínimo de muestras requeridas en cada nodo hoja
min_samples_leaf = [1, 2, 4]
# Método de selección de muestras para entrenar cada árbol
bootstrap = [True]

param_grid = dict (n_estimators = n_estimators, max_features = max_features,
               max_depth = max_depth, min_samples_split = min_samples_split,
               min_samples_leaf = min_samples_leaf, bootstrap = bootstrap)

#SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)
```

```
grid_search = GridSearchCV(rf, param_grid = param_grid, scoring='accuracy',
                           cv = kf, return_train_score=True)


#CLASIFICADOR ÓPTIMO

grid_search.fit(X_r, y_train)
best_model = grid_search.best_estimator_

values = grid_search.best_params_

print('Mejor precisión de entrenamiento: {0:0.4f}, con parámetros = {1}'
      .format(grid_search.best_score_, grid_search.best_params_) + '\n')

#VALIDACIÓN

X_s = scaler.transform(X_test)
X_test = pca.transform(X_s)

best_score = grid_search.score(X_test, y_test)

print ('Precisión de validación del mejor modelo obtenido:{0:0.4f}'
       .format(best_score))
```

10.2.6 RandomForest_GRL.py

Ajuste de los hiperparámetros del clasificador Random Forest aplicando una búsqueda de rejilla con validación cruzada (GridSearchCV) con el vector de características GRL.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler
```

```
#CONJUNTO DE DATOS

filename = 'DataBase_GRL'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()


X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#ESCALADO DEL VECTOR DE CARACTERÍSTICAS

scaler = MinMaxScaler()
X_s = scaler.fit_transform(X_train)

#MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

pca = PCA(n_components=6)
X_r = pca.fit_transform(X_s)

#ESTIMADOR

rf = RandomForestClassifier(random_state = 42)

#CUADRÍCULA DE PARÁMETROS

# Número de árboles en random forest
n_estimators = [100, 400, 800, 1000]
# Número de características a tener en cuenta en cada división
max_features = ['auto', 'sqrt']
# Número máximo de niveles en los árboles
max_depth = [50, 60, 70]
max_depth.append(None)
# Número mínimo de muestras requeridas en cada nodo
min_samples_split = [2, 5, 10]
# Número mínimo de muestras requeridas en cada nodo hoja
min_samples_leaf = [1, 2, 4]
# Método de selección de muestras para entrenar cada árbol
bootstrap = [True]

param_grid = dict (n_estimators = n_estimators, max_features = max_features,
               max_depth = max_depth, min_samples_split = min_samples_split,
               min_samples_leaf = min_samples_leaf, bootstrap = bootstrap)

#SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

grid_search = GridSearchCV(rf, param_grid = param_grid, scoring='accuracy',
                           cv = kf, return_train_score=True)
```

```
#CLASIFICADOR ÓPTIMO

grid_search.fit(X_r, y_train)
best_model = grid_search.best_estimator_

values = grid_search.best_params_

print('Mejor precisión de entrenamiento: {0:0.4f}, con parámetros = {1}'
      .format(grid_search.best_score_, grid_search.best_params_) + '\n')
#VALIDACIÓN

X_s = scaler.transform(X_test)
X_test = pca.transform(X_s)

best_score = grid_search.score(X_test, y_test)

print ('Precisión de validación del mejor modelo obtenido:{0:0.4f}'
       .format(best_score))
```

10.2.7 LogisticRegression.py

Ajuste del hiperparámetro C de la regresión logística multinomial aplicando validación cruzada anidada. El valor óptimo se obtiene con una búsqueda de rejilla (GridSearchCV).

```
#CONFIGURACIÓN DEL ENTORNO

import pickle
import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab' #Modificar para cada vector de características
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()
```

```
X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

scaler = StandardScaler()
pca = PCA(n_components=6)

#ESTIMADOR Y PARÁMETRO

lr = LogisticRegression(multi_class='multinomial', solver='lbfgs',
                        penalty='l2', max_iter = 1000)

#Valores que adoptará el hiperparámetro C
param_C = np.logspace(-4, 1, 6)

param_grid = dict (C = param_C)

#Validación cruzada k-Fold
kf = KFold(n_splits=5, shuffle=True, random_state = 42)

grid = GridSearchCV(lr, param_grid= param_grid,scoring='accuracy', cv=kf,
                    return_train_score=True)

#CLASIFICADOR ÓPTIMO

it=1 #Contador de iteraciones

values = [] #Vector en el que se almacena el valor óptimo de la iteración
test_scores = [] #Vector en el que se almacena el score del conjunto de
validación de cada iteración


for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    X_r = pca.fit_transform(X_s)

    #GRID SEARCH

    grid.fit(X_r,y_train)

    tr_mean = grid.cv_results_['mean_train_score']
    tr_std = grid.cv_results_['std_train_score']
```

```
    cv_mean = grid.cv_results_['mean_test_score']
    cv_std = grid.cv_results_['std_test_score']

    params = grid.cv_results_['params']



    plt.semilogx(param_C, tr_mean, linestyle="-", color="darkorange",
                 label="training score")
    plt.semilogx(param_C, cv_mean, linestyle="-", color="b",
                 label="cross-validation score")

    lw = 2
    plt.fill_between(param_C, tr_mean - tr_std, tr_mean + tr_std, alpha=0.2,
                 color="darkorange", lw=lw)
    plt.fill_between(param_C, cv_mean - cv_std, cv_mean + cv_std, alpha=0.2,
                 color="b", lw=lw)

    title = 'Grid Search. Iteración ' + str(it)
    plt.title(title)
    plt.xlabel("C")
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    plt.legend(loc="best")

    plt.savefig('Results/LogRegression_GS_it'+ str(it) + '.png')
    plt.show()

    value = grid.best_params_['C']
    values.append(value)
    print('\n' + 'El valor óptimo de C con Grid Search es: ', value)
    print('Mejor precisión alcanzada: {0:0.4f}, con C = {1}'
          .format(grid.best_score_, grid.best_params_['C']))

    #COMPROBACIÓN
    X_s = scaler.fit_transform(X_test)
    X_test = pca.fit_transform(X_s)

    test_score = grid.score(X_test, y_test)
    test_scores.append(test_score)
    print ('\n' + 'Precisión de la validación con valor óptimo: {:0.4f}'
           .format(test_score) + '\n')


    it+=1

test_mean = np.mean(test_scores)
test_std = np.std(test_scores)

print ('Precisión media de la validación: {:0.4f}'.format(test_mean))
print ('Desviación típica de la precisión de validación: {:0.4f}'
       .format(test_std))
```

10.3 Anexo III: Ficheros relacionados con el clasificador óptimo y la interfaz gráfica de usuario.

10.3.1 OptimalClassifier_HSab_inicial.py

Comparación de clasificadores para obtener el clasificador óptimo para el vector de características HSab Inicial.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.svm import SVC

from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab_Inicial'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#CLASIFIACADORES CON HIPERPARÁMETROS AJUSTADOS

rbf = SVC(C = 100, gamma = 0.01, random_state = 42)

#CLASIFICADOR ÓPTIMO

#Validación cruzada k-Fold
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del conjunto
de validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración
```

```
for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    scaler = MinMaxScaler()
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    pca = PCA(n_components=6)
    X_r = pca.fit_transform(X_s)

    #ENTRENAMIENTO DEL CLASIFICADOR

    rbf.fit(X_r, y_train)

    #VALIDACIÓN

    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)


    test_score = rbf.score(X_test, y_test)
    test_scores.append(test_score)

    coef = rbf.support_vectors_
    coefs.append(coef)

    print('\n' + 'Precisión en el conjunto de validación: {0:0.4f}'
          .format(test_score))

    it+=1
```

10.3.2 OptimalClassifier_HSab.py

Comparación de clasificadores para obtener el clasificador óptimo para el vector de características HSab.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#CLASIFIACADORES CON HIPERPARÁMETROS AJUSTADOS

rf = RandomForestClassifier(n_estimators = 1400, min_samples_split = 2,
                            min_samples_leaf = 1, max_features ='auto',
                            max_depth=10, bootstrap=True,random_state=42)

#CLASIFICADOR ÓPTIMO

#Validación cruzada k-Fold
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del conjunto
de validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    scaler = MinMaxScaler()
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    pca = PCA(n_components=6)
    X_r = pca.fit_transform(X_s)

    #ENTRENAMIENTO DEL CLASIFICADOR

    rf.fit(X_r, y_train)
```

```
    #VALIDACIÓN

    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)


    test_score = rf.score(X_test, y_test)
    test_scores.append(test_score)

    coef = rf.feature_importances_
    coefs.append(coef)

    print('\n' + 'Precisión en el conjunto de validación: {0:0.4f}'
          .format(test_score))

    it+=1
```

10.3.3 OotimalClassifier_GRLab.py

Comparación de clasificadores para obtener el clasificador óptimo para el vector de características GRLab.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression

from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_GRLab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names
```

```
#CLASIFIACADORES CON HIPERPARÁMETROS AJUSTADOS

linear = LinearSVC(C = 0.01, random_state = 42)
rbf = SVC(C = 10, gamma = 0.01, random_state = 42)
polynomial = SVC(C = 1, kernel ='poly', degree = 2, random_state = 42)
lr = LogisticRegression( C = 10, multi_class='multinomial', solver='lbfgs',
                         penalty='l2', max_iter = 1000)

#CLASIFICADOR ÓPTIMO

#Validación cruzada k-Fold
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del conjunto de
validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    scaler = MinMaxScaler()
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    pca = PCA(n_components=6)
    X_r = pca.fit_transform(X_s)

    #ENTRENAMIENTO DEL CLASIFICADOR

    linear.fit(X_r, y_train)
    rbf.fit(X_r, y_train)
    polynomial.fit(X_r, y_train)
    lr.fit(X_r, y_train)

    #VALIDACIÓN

    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    linear_score = linear.score(X_test, y_test)
    linear_coef = linear.coef_

    rbf_score = rbf.score(X_test, y_test)
    rbf_coef = rbf.support_vectors_
```

```
    polynomial_score = polynomial.score(X_test, y_test)
    polynomial_coef = polynomial.support_vectors_

    lr_score = lr.score(X_test, y_test)
    lr_coef = lr.coef_


    test_score = {'linear': linear_score, 'rbf': rbf_score,
                  'polynomial': polynomial_score,
                  'logistic regression': lr_score}
    test_scores.append(test_score)

    coef = {'linear': linear_coef, 'rbf': rbf_coef,
            'polynomial': polynomial_coef,
            'logistic regression': lr_coef}
    coefs.append(coef)

    print('\n' + 'Precisión en el conjunto de validación:')

    [print('{0}: {1:0.4f}'.format(key,test_score[key])) for key in test_score]


    it+=1
```

10.3.4 OptimalClassifier_GRL.py

Comparación de clasificadores para obtener el clasificador óptimo para el vector de características GRL.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_GRL'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()
```

```
X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#CLASIFIACADORES CON HIPERPARÁMETROS AJUSTADOS

linear = LinearSVC(C = 0.01, random_state = 42)
polynomial = SVC(C = 10, kernel ='poly', degree = 2, random_state = 42)
lr = LogisticRegression( C = 0.01, multi_class='multinomial', solver='lbfgs',
                         penalty='l2', max_iter = 1000)
rf = RandomForestClassifier(n_estimators = 100, min_samples_split = 10,
                            min_samples_leaf = 4, max_features ='auto',
                            max_depth = 50,bootstrap = True,random_state = 42)

#CLASIFICADOR ÓPTIMO

#Validación cruzada k-Fold
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del conjunto de
validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    scaler = MinMaxScaler()
    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    pca = PCA(n_components=6)
    X_r = pca.fit_transform(X_s)

    #ENTRENAMIENTO DEL CLASIFICADOR

    linear.fit(X_r, y_train)
    polynomial.fit(X_r, y_train)
    lr.fit(X_r, y_train)
    rf.fit(X_r, y_train)
```

```
    #VALIDACIÓN

    X_s = scaler.transform(X_test)
    X_test = pca.transform(X_s)

    linear_score = linear.score(X_test, y_test)
    linear_coef = linear.coef_

    polynomial_score = polynomial.score(X_test, y_test)
    polynomial_coef = polynomial.support_vectors_

    lr_score = lr.score(X_test, y_test)
    lr_coef = lr.coef_

    rf_score = rf.score(X_test, y_test)
    rf_coef = rf.feature_importances_

    test_score = {'linear': linear_score, 'polynomial': polynomial_score,
                  'logistic regression': lr_score, 'random forest': rf_score}
    test_scores.append(test_score)

    coef = {'linear': linear_coef,  'polynomial': polynomial_coef,
                  'logistic regression': lr_coef, 'random forest': rf_coef}
    coefs.append(coef)

    print('\n' + 'Precisión en el conjunto de validación:')

    [print('{0}: {1:0.4f}'.format(key,test_score[key])) for key in test_score]


    it+=1
```

10.3.5 Classifier_RandomForest.py

Mejoras del clasificador Random Forest con vector de características HSab para aumentar la precisión.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#CLASIFIACADOR
clf = RandomForestClassifier(n_estimators = 1600, min_samples_split = 10,
                        min_samples_leaf = 1, max_features ='auto',
                        max_depth = 10, bootstrap = True, random_state = 42)

#MÉTODOS DE ESCALADO Y EXTRACCIÓN DE CARACTERÍSTICAS
scaler = MinMaxScaler()
pca=PCA()
kpca = KernelPCA(kernel ='rbf', gamma = 0.1)

#VALIDACIÓN CRUZADA
loo = LeaveOneOut()
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del conjunto de
validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración
```

```
for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    #ESCALADO DEL VECTOR DE CARACTERÍSTICAS

    X_s = scaler.fit_transform(X_train)

    #MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

    X_r = kpca.fit_transform(X_s)

    #ENTRENAMIENTO DEL CLASIFICADOR

    clf.fit(X_r, y_train)

    #VALIDACIÓN

    X_s = scaler.transform(X_test)
    X_test = kpca.transform(X_s)


    test_score = clf.score(X_test, y_test)
    test_scores.append(test_score)

    coef = clf.feature_importances_
    coefs.append(coef)

    print('\n' + 'Precisión en el conjunto de validación: {0:0.4f}'
          .format(test_score))


    it+=1
```

10.3.6 Classifier_SVM.py

Mejoras del clasificador SVM con *kernel* RBF con vector de características GRLab para aumentar la precisión.

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_GRLab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target
target_names = DataBase.target_names

#CLASIFIACADOR
clf = SVC(C = 100, gamma = 0.1, random_state = 42)

#MÉTODOS DE ESCALADO Y EXTRACCIÓN DE CARACTERÍSTICAS
scaler = MinMaxScaler()
pca = PCA()
kpca = KernelPCA( kernel='rbf', gamma = 0.001)

#VALIDACIÓN CRUZADA
loo = LeaveOneOut()
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

it=1 #Contador de iteraciones

test_scores = [] #Vector en el que se almacena la precisión del
conjunto de validación de cada iteración
coefs = [] #Vector en el que se almacenan los pesos asignados a las
características en cada iteración

for train_index, test_index in kf.split(X):

    #CONJUNTO DE DATOS: SUBCONJUNTOS DE ENTRENAMIENTO Y VALIDACIÓN

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
#ESCALADO DEL VECTOR DE CARACTERÍSTICAS

X_s = scaler.fit_transform(X_train)

#MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS

X_r = pca.fit_transform(X_s)

#ENTRENAMIENTO DEL CLASIFICADOR

clf.fit(X_r, y_train)

#VALIDACIÓN

X_s = scaler.transform(X_test)
X_test = pca.transform(X_s)


test_score = clf.score(X_test, y_test)
test_scores.append(test_score)

coef = clf.support_vectors_
coefs.append(coef)

print('\n' + 'Precisión en el conjunto de validación: {0:0.4f}'
      .format(test_score))


it+=1
```

10.3.7 Classifier.py

```
#CONFIGURACIÓN DEL ENTORNO

import pickle

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import MinMaxScaler

#CONJUNTO DE DATOS

filename = 'DataBase_HSab'
infile = open(filename,'rb')

DataBase = pickle.load(infile)
infile.close()

X = DataBase.data
y = DataBase.target

#CLASIFIACADOR

clf = RandomForestClassifier(n_estimators = 1600, min_samples_split
= 10,
                             min_samples_leaf = 1, max_features
='auto',
                             max_depth = 10, bootstrap = True,
random_state = 42)

#ESCALADO DEL VECTOR DE CARACTERÍSTICAS

scaler = MinMaxScaler()
X_s = scaler.fit_transform(X)

#MÉTODO DE EXTRACCIÓN DE CARACTERÍSTICAS PCA

pca = KernelPCA(kernel ='rbf', gamma = 0.1)
X_r = pca.fit_transform(X_s)

#ENTRENAMIENTO DEL CLASIFICADOR

clf.fit(X_r, y)

#EXPORTAR CLASIFICADOR

filename = 'classifier'
outfile = open(filename,'wb')

pickle.dump(clf,outfile)
outfile.close()
```

```
#EXPORTAR PCA

filename = 'PCA'
outfile = open(filename,'wb')

pickle.dump(pca,outfile)
outfile.close()

#EXPORTAR ESCALADOR

filename = 'Scaler'
outfile = open(filename,'wb')

pickle.dump(scaler,outfile)

outfile.close()
```

10.3.8 GUI.py

Interfaz gráfica para la clasificación de muestras de aceite de oliva en función de su contenido en impurezas no solubles. Este fichero contiene la configuración de la interfaz gráfica, así como las funciones auxiliares necesarias para su funcionamiento.

```
#CONFIGURACIÓN DEL ENTORNO

import cv2
import time
import pickle
import numpy as np
import streamlit as st

from PIL import Image
from vector import hist_agrupado
from vector import img_segmentation
```

```
def load_model(path):

    """Carga el modelo entrenado

    Parámetro de entrada:
        - path: str. Ubicación del archivo."""

    filename = path + '\classifier'
    infile = open(filename,'rb')
    clf = pickle.load(infile)
    infile.close()

    return clf


def input_vector(image):

    """Elabora el vector de características a partir de una imagen
    proporcionada.

    Parámetro de entrada:
        - image: array of uint8. Imagen de muestra."""

    imageBGR = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) #PIL carga la imagen
como RGB
    resized_img = cv2.resize(imageBGR,(1600,1200))
    sample = img_segmentation(resized_img,(0,210,50),(40,255,255),7)

    n = 4 #Factor de agrupación


    #Histogramas que componen el vector de características:
    G = hist_agrupado(sample,'BGR',1,n)
    R = hist_agrupado(sample,'BGR',2,n)
    L = hist_agrupado(sample,'CIELAB',0,n)
    a = hist_agrupado(sample,'CIELAB',1,n)
    b = hist_agrupado(sample,'CIELAB',2,n)

    vector = G+R+L+a+b #Se juntan los histogramas en un solo vector
    vector = np.array(vector, int)
    vector = vector.reshape(1, -1)

    return vector
```

```
def image_preprocessing(image, path):

    """Preprocesamiento de los datos de entrada al clasificador. Emplea la
    función auxiliar input_vector para obtener el vector de características
    a partir de la imagen de muestra proporcionada.

    Parámetro de entrada:
        - image: array of uint8. Imagen de muestra
        - path: str. Ubicación del archivo."""

    filename_pca = path + '\PCA'
    infile = open(filename_pca,'rb')
    pca = pickle.load(infile)
    infile.close()

    filename = path + '\Scaler'
    infile = open(filename,'rb')
    scaler = pickle.load(infile)
    infile.close()

    #OBTENCIÓN DEL VECTOR DE CARACTERÍSTICAS
    vector = input_vector(image)

    #NORMALIZACIÓN
    X_s = scaler.transform(vector)

    #EXTRACCIÓN DE CARACTERÍSTICAS
    X_r = pca.transform(X_s)

    return X_r


def model_prediction(X_r, clf):

    """Realiza la predicción del nivel de impurezas de una muestra de aceite
    de oliva a partir de su vector de características y un clasificador dado.

    Parámetros de entrada:
       - X_r: array of float64. Vector de características de la imagen de
       muestra.
       - clf: svm._classes.SVC. Clasificador entrenado"""

    labels = ['BAJO', 'MEDIO', 'ALTO']
    prediction = clf.predict(X_r)

    label = labels[prediction[0]]

    return label
```

```
def main():

    #TÍTULO Y ENCABEZADO
    st.title('CONTENIDO DE IMPUREZAS INSOLUBLES EN MUESTRAS DE ACEITE DE
OLIVA')

    #UBICACIÓN DEL MODELO ENTRENADO
    st.header('ENTRADA DE DATOS')
    st.subheader('Ubicación de los archivos')

    path = st.text_input('Introduzca la ubicación del clasifcador:', '')

    if path == '':
        st.warning("Por favor, introduzca una ubicación.")
        st.stop()

    else:

        #IMAGEN DE MUESTRA
        st.subheader('Imagen de muestra')
        uploaded_file = st.file_uploader(label="Seleccione la imagen de
                    una muestra de aceite:",type=["png", "jpeg", "jpg"])

        if not uploaded_file:
            st.warning("Por favor, suba una imagen.")
            st.stop()
        else:

            image = np.array(Image.open(uploaded_file))

            #MOSTRAR IMAGEN CARGADA
            st.image(image, caption = 'Imagen analizada',
                     use_column_width=True)



            #REALIZAR CLASIFICACIÓN

            st. header ('CLASIFICACIÓN DE LA MUESTRA')
            st.write('Presione IDENTIFICAR para realizar la estimación')

            if st.button('IDENTIFICAR'):

                with st.spinner('Realizando clasificación ...'):

                    clf = load_model(path)

                    X_r = image_preprocessing(image, path)

                    prediction = model_prediction(X_r, clf)

                    time.sleep(2)
                    st.success('¡Clasificación completada!')
```

```
                time.sleep(0.5)
                st.subheader('RESULTADO DE LA PREDICCIÓN')
                time.sleep(0.5)
                st.success('Nivel de impurezas insolubles estimado: {0}'
                           .format(prediction))


if __name__ == '__main__':
    main()
```