República del Ecuador
Universidad Tecnológica Empresarial de Guayaquil - UTEG

Trabajo de Titulación
para la obtención del título de:
Ingeniero en Software

Tema:
Machine learning aplicado a la clasificación y optimización de quimiotipos en aceites esenciales de cymbopogon citratus (hierbaluisa)

Autor/a:
Borys Álvaro Cereceda Guzmán

Director de trabajo de titulación:
Ing. Grace Katiuska Viteri Guzman. MSc.

Septiembre 2024

Guayaquil - Ecuador

AGRADECIMIENTO

Quisiera expresar mi más sincero agradecimiento a los profesores que me acompañaron y guiaron durante el desarrollo de esta tesis. A la profesora. Grace Viteri, por su invaluable orientación y su paciencia en cada etapa de este proceso; al profesor Sedolfo Carrasquero, por sus conocimientos y apoyo constante, y al profesor Julio Gonzales, por sus consejos y compromiso con mi crecimiento profesional. Su experiencia y dedicación fueron esenciales para la realización de este trabajo y para mi formación académica.

Extiendo mi gratitud a la Universidad Tecnológica Empresarial de Guayaquil por brindarme los recursos y el entorno necesario para el desarrollo de este proyecto, así como a mis compañeros y amigos, que hicieron este camino más ameno y compartieron sus conocimientos conmigo.

DEDICATORIA

A mi madre, por ser las columnas de mi templo.

Por enseñarme con su amor y con su ejemplo.

Gracias por todo tu esfuerzo y sacrificio, quiero que sepas que cada granito de arena aportado no ha sido en vano, espero que donde sea que te encuentres te sientas orgullosa y sepas lo mucho que te amo.

A mi hermano, por ser calor y abrigo, a aconsejarme cuando busco un buen amigo

Por perdonarme cuando no se lo que digo y por estar siempre conmigo como el sol esta en el trigo

Va por ti y por las piedras del camino.

A mi novia, por ser ese apoyo incondicional, por entender el valor de estar en las buenas y en las malas, por hacerme entender que a todo se te tiene que poner ganas que las cosas no siempre salen bien y cuando salen bien agradecer, gracias por estar, te amo.

A mi familia, mis tíos, y mis primos, gracias por ser esa familia incondicional porque cuando mas los necesite siempre estuvieron ahí, gracias por todo el apoyo brindado.

A mi amigo Mario, por comprender el valor de la amistad, a enseñarme a no rendirme y seguir intentándolo.

A mis amigos, por todo su apoyo incondicional.

A mis gatos, sin ellos nada de esto hubiera resultado, gracias por acompañarme en esas madrugadas donde por momentos resultaba imposible continuar.

A todo aquel que formo parte de este proceso quedo eternamente agradecido.

DECLARACIÓN DE AUTORÍA

Yo, Borys Alvaro Cereceda Guzmán, con número de cédula 0953496676, declaro que el trabajo titulado “Machine Learning aplicado a la clasificación y optimización de quimiotipos en aceites esenciales de Cymbopogon citratus (Hierbaluisa)” es de mi autoría. Afirmo que la información, los análisis, y los resultados presentados en este documento son fruto de mi esfuerzo y dedicación. Además, confirmo que he dado el crédito correspondiente a todas las fuentes consultadas y que este trabajo es completamente original, en cumplimiento con las normativas de propiedad intelectual y ética académica establecidas por la Universidad Tecnológica Empresarial de Guayaquil.

Con esta declaración, asumo la responsabilidad total de la veracidad y autenticidad del contenido aquí presentado.

TÍTULO DEL ARTÍCULO

MACHINE LEARNING APLICADO A LA CLASIFICACIÓN Y OPTIMIZACIÓN DE QUIMIOTIPOS EN ACEITES ESENCIALES DE CYMBOPOGON CITRATUS (HIERBALUISA)

Borys Álvaro Cereceda Guzmán
boryscereceda@hotmail.com

RESUMEN

Este trabajo se centra en la aplicación de técnicas de Machine Learning para la clasificación y optimización de quimiotipos en aceites esenciales de Cymbopogon citratus (Hierbaluisa). El objetivo principal fue desarrollar un modelo que permita mejorar la precisión en la identificación de las composiciones químicas de los aceites esenciales, utilizando algoritmos como la regresión lineal y el Random Forest. Se aplicaron técnicas de preprocesamiento de datos y análisis de componentes principales (PCA) para la reducción de la dimensionalidad y la normalización de los datos espectrales. Los resultados demostraron que el modelo Random Forest superó al de regresión lineal, obteniendo un menor error cuadrático medio (MSE) y un mejor coeficiente de determinación (R^2), lo que indica una mayor precisión en la clasificación. El estudio concluye que el uso de Machine Learning es eficaz para la optimización de quimiotipos en aceites esenciales de Cymbopogon citratus, permitiendo la identificación de patrones complejos en los datos.

Palabras clave: Regresión lineal, Randon Forest, análisis de componentes principales (PCA), Error Cuadratico medio, Coeficiente de determinación.

INTRODUCCIÓN

En el campo de la ingeniería de software, una de las principales problemáticas es el desarrollo de sistemas que puedan procesar y analizar grandes volúmenes de datos de manera eficiente para extraer información valiosa y accionable. En particular, el procesamiento de datos complejos e incompletos es una barrera significativa en la industria de los aceites esenciales, donde la clasificación y optimización de quimiotipos son esenciales para el desarrollo de productos distintivos y de alta calidad.

Los métodos tradicionales de análisis químico a menudo requieren datos completos y bien estructurados, lo que no siempre es posible en la práctica debido a la variabilidad y complejidad de las muestras de Cymbopogon citratus, comúnmente conocido como hierbaluisa, es una planta con un gran potencial en la producción de aceites esenciales. Esta situación demanda soluciones innovadoras que no solo manejen datos incompletos, sino que también proporcionen predicciones precisas y útiles. Este proyecto busca abordar estos desafíos mediante el uso de técnicas avanzadas de Machine Learning (ML) y un enfoque de desarrollo de software orientado al modularidad y escalabilidad, proporcionando una herramienta robusta y eficiente para la clasificación y optimización de aceites esenciales de Cymbopogon Citratus.

¿Cómo puede un algoritmo de predicción basado en Machine Learning, utilizando regresión lineal de gradiente descendiente y Análisis de Componentes Principales (PCA), mejorar la clasificación de los quimiotipos de aceites esenciales de Cymbopogon citratus a partir de datos incompletos?

Hoy en día, la inteligencia artificial ha revolucionado diversos campos, demostrando ser una herramienta poderosa y versátil capaz de ofrecer soluciones innovadoras y eficientes en múltiples situaciones. Desde la medicina hasta la agricultura, la inteligencia artificial está transformando la manera en que abordamos problemas complejos, permitiendo avances significativos que antes eran impensables.

La presente investigación dará un enfoque de como el Machine Learning (ML) puede ser una herramienta de la inteligencia artificial, capaz de transformar grandes cantidades de datos en información útil y accionable.

OBJETIVOS

Objetivo general

Desarrollar un modelo de Machine Learning utilizando múltiples algoritmos y técnicas de clasificación para optimizar la precisión en la clasificación de quimiotipos de aceites esenciales de Cymbopogon citratus (Hierbaluisa) y mejorar la compresión de las relaciones entre sus variables.

Objetivos específicos

- Preprocesar un conjunto de datos de aceites esenciales de Cymbogon citratus (Hierbaluisa).
- Comparar el desempeño del modelo propuesto con otros algoritmos de clasificación populares.
- Visualizar los resultados del modelo para una compresión de las relaciones entre las variables y las clases de quimiotipos.

MARCO TEÓRICO

En la última década, las herramientas de Machine Learning han demostrado ser muy útiles para analizar grandes volúmenes de datos y extraer información relevante. En el campo de los aceites esenciales, se han realizado estudios para explorar el potencial de estas técnicas en la clasificación y optimización de productos.

“Sarker et al. (2021) comentan que la inteligencia artificial facilita el procesamiento de grandes volúmenes de datos, permitiendo la aplicación de técnicas más avanzadas y eficientes para manejar información compleja. Según su estudio, la integración de IA no solo automatiza ciertas tareas, sino que también permite obtener insights valiosos que serían difíciles de alcanzar mediante métodos tradicionales.”

"Ullah et al. (2021) demostraron que los modelos de aprendizaje automático, como el Random Forest, son capaces de realizar predicciones bastante precisas sobre el rendimiento del bio-oil. Su estudio concluyó que estos modelos superan a los métodos tradicionales, especialmente la regresión multilineal, en la gestión de la complejidad y la variabilidad de los datos."

“Al-Hadi et al. (2022) destacaron el creciente uso de sistemas de clasificación que se basan en modelos matemáticos, como la cromatografía de gases acoplada a espectrometría de masas (GC-MS). Su investigación resalta la eficiencia de algoritmos de redes neuronales artificiales y de k-vecinos más cercanos para clasificar con precisión compuestos químicos, subrayando cómo estas tecnologías están transformando las prácticas analíticas en áreas como la química de los aceites esenciales."

están transformando las prácticas analíticas en campos como la química de los aceites esenciales.

Estos estudios previos respaldan la idea de que el Machine Learning puede ser una herramienta eficaz para abordar los desafíos de clasificación en el análisis de aceites esenciales. Al igual que en estos trabajos, nuestra investigación se centrará en aplicar técnicas de Machine Learning para mejorar la clasificación de quimiotipos en aceites esenciales de Cymbopogon citratus.

Machine Learning: Una herramienta para la exploración de datos

De acuerdo con S. Brown (2021). Machine Learning es una rama de la inteligencia artificial que permite a las computadoras aprender y mejorar a partir de la experiencia sin ser explícitamente programadas para ello. A través de algoritmos y modelos matemáticos, las máquinas pueden identificar patrones en los datos y realizar predicciones o decisiones basadas en esos patrones.

Según Jordan & Mitchell (2015) El Machine Learning (ML) se ha convertido en una herramienta fundamental para la exploración de datos debido a su capacidad para procesar y analizar grandes volúmenes de información de manera eficiente y precisa. A diferencia de los métodos tradicionales de análisis de datos, que a menudo requieren una programación específica y un entendimiento detallado de las relaciones entre las variables, los algoritmos de ML pueden identificar patrones complejos y hacer predicciones basadas en datos existentes sin intervención humana directa.

Como plantea Hinton & Salakhutdinov (2006) Uno de los principales beneficios de utilizar ML en la exploración de datos es su capacidad para manejar datos de alta dimensionalidad y descubrir relaciones ocultas que no son evidentes a simple vista. Por ejemplo, técnicas como el Análisis de Componentes Principales (PCA) permiten reducir la dimensionalidad de los datos, facilitando la identificación de las características más relevantes y la visualización de estructuras subyacentes en conjuntos de datos complejos.

Lebanov & Pagán (2021) Considera que esta técnica es especialmente útil en campos como

la química de los aceites esenciales, donde las muestras pueden contener múltiples compuestos y variaciones que afectan su clasificación y propiedades.

Sarker et al. (2021) Define que el aprendizaje supervisado y no supervisado son dos enfoques clave en ML que se utilizan para la exploración de datos. En el aprendizaje supervisado, los modelos son entrenados con datos etiquetados para aprender a predecir resultados específicos, mientras que, en el aprendizaje no supervisado, los algoritmos identifican patrones y agrupamientos en los datos sin necesidad de etiquetas previas.

Viciano-Tudela et al., (2023) Señala que, además la capacidad de ML para automatizar el análisis de datos ha demostrado ser un avance significativo en la investigación científica y la toma de decisiones empresariales. Los algoritmos de ML pueden procesar rápidamente grandes volúmenes de datos para proporcionar información accionable, lo que permite a los investigadores y profesionales tomar decisiones basadas en datos más rápidamente que con los métodos tradicionales.

Aprendizaje supervisado

Según IBM (2021) El aprendizaje supervisado es una técnica de Machine Learning en la que un algoritmo se entrena utilizando un conjunto de datos etiquetados. En estos datos, cada ejemplo de entrada está asociado con una clase o valor objetivo específico, lo que permite al modelo aprender a mapear las características de entrada con las salidas deseadas.

Este proceso de entrenamiento busca optimizar la precisión del modelo para que pueda realizar predicciones precisas sobre nuevos datos. En el contexto de la clasificación de quimiotipos de aceites esenciales, se pueden usar modelos supervisados para identificar y clasificar correctamente las diferentes composiciones químicas presentes en las muestras de Cymbopogon citratus (Hierbaluisa).

Automatizando la identificación de compuestos

H2O.ai (2024) Describe que los algoritmos de Machine Learning, especialmente los supervisados, son capaces de reconocer patrones complejos en los datos espectrales y asociarlos con compuestos químicos específicos. Utilizando técnicas como los bosques aleatorios (Random Forest) y los clasificadores de k-vecinos más cercanos (k-NN), los modelos pueden identificar de manera precisa la presencia de ciertos compuestos obtenidos en la similitud con los ejemplos de entrenamiento.

Optimizado a la extracción de características

De acuerdo con Built In (2024), PCA permite identificar y extraer las características más relevantes de un conjunto de datos, reduciendo su complejidad sin perder información crucial para la clasificación. Esto facilita la visualización y el análisis de las relaciones subyacentes en los datos, optimizando la precisión de los modelos al centrado solo en las variables más significativas.

Mejorado a la precisión de la clasificación

IBM (2021) Manifiesta que los modelos de clasificación basados en Machine Learning, como los de regresión logística y los bosques aleatorios, han demostrado ser más precisos en la identificación de quimiotipos en comparación con los métodos tradicionales. Estos modelos pueden aprender de los errores durante el proceso de entrenamiento y ajustar sus parámetros para mejorar continuamente su capacidad de clasificación.

En el estudio de los aceites esenciales de *Cymbopogon citratus*, estas técnicas permiten una identificación más exacta de las variaciones químicas, lo cual es esencial para la optimizacióndel producto.

METODOLOGÍA

La presente investigación sigue un enfoque cuantitativo, de diseño no experimental de los datos obtenidos sobre los aceites esenciales de Cymbopogon citratus (hierbaluisa)**,** donde su alcance será descriptivo a través del preprocesamiento de datos y la visualización de los resultados, se detalla las características de las variables y se describen las relaciones identificadas entre ellas.

Técnicas de recolección de datos, población y muestra

Los métodos utilizados siguen las metodologías descritas en varios trabajos de titulación, entre estos trabajos se destaca el realizado por Araujo et al. (2017), donde la población se compone por todos los tipos de aceites esenciales, y la muestra de datos específicos que representan estos diferentes tipos.

Google Colab: Se utilizó esta plataforma de programación en la nube para desarrollar, ejecutar y probar los scripts de Machine Learning. Google Colab facilitó la ejecución de los algoritmos de clasificación, proporcionando un entorno interactivo y acceso a recursos computacionales como GPU para acelerar los cálculos.

Python y Librerias de Machine Learning

Pandas: Utilizada para la manipulación, limpieza y análisis de datos. Facilito la preparación de los datos antes de su ingreso a los modelos de Machine Learning.

Scikit-Learn: Esta librería fue fundamental para implementar diversos algoritmos de clasificación

PCA (Principal Component Analysis): Técnica de reducción de dimensionalidad utilizada para identificar y seleccionar las características más relevantes del conjunto de datos, mejorando así la eficiencia y la precisión del modelo de clasificación.

RandomForest: Algoritmo de clasificación utilizado para manejar grandes volúmenes de

datos con alta dimensionalidad. Este método permitió crear un modelo robusto que mejora la precisión en la clasificación de los quimiotipos de aceites esenciales.

LinearRegression: Empleado para construir modelos predictivos que analicen la relación entre variables dependientes e independientes. Aunque no es el enfoque principal de clasificación, ayudó a entender la correlación lineal entre ciertas características.

Matplotlib y Seaborn: Herramientas de visualización de datos utilizadas para crear gráficos que representen las relaciones entre las variables y los resultados de la clasificación de quimiotipos.

Microsoft Excel: Usado para la organización y revisión inicial de los datos del archivo. Excel facilito una inspección visual preliminar de los datos, permitiendo identificar valores atípicos.

RESULTADOS

Preprocesamiento de los datos

Dado que el DataFrame contiene valores faltantes o NaN, para poder reemplazar estos valores con la media se realizó una imputación de los datos NaN, con la clase ‘SimpleImputer’, esto es crucial para asegurar que los algoritmos de ML puedan procesar los datos sin errores (Tabla 1).

Tabla 1

DataFrame con valores imputados

WL	C/G	G/N	P/G
AGUACLARA	0.000	1.068	0.0398
AGUAL02	0.000	0.821	0.6335
ARCHT	0.0001	11.932	0.4353
ASUS	0.0143	0.9155	0.3762
BALREP	0.000	10.621	0.0297
BASE	0.0143	0.9155	0.3762
BB	0.0143	0.9155	0.3762
CAT01	0.000	0.7846	0.1930
CAT02	0.000	0.8006	0.4708
CHANK	0.0143	0.9155	0.3762
CHE	0.0143	0.9155	0.3762

Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Modelado y Predicción

Se ha creado dos clases ‘ModelW’ y ‘ModelRandomForest’, que encapsula la lógica para entrenar los modelos, realizar predicciones, guardar resultados y visualizar datos.

La clase ‘ModelW’ se implementó con el modelo matemático ‘LinearRegression’ que nos ayuda a predecir el valor de datos desconocidos (Tabla 2).

Por otro lado, la clase ‘ModelRandomForest’, se utilizó el algoritmo de ML ‘RandomForest’ que combina las predicciones de múltiples arboles de decisión para producir una predicción más precisa y estable (Tabla 3).

El uso de estos algoritmos es para llevar a cabo una comparación de cual resulta más eficiente para la predicción de datos.

Tabla 2

LinearRegression

WL	**STATE**	**C/G**	**G/N**	**P/G**
AGUACLARA	original	0.000000	1068000	0.039800
AGUAL02	original	0.000000	0.820700	0.633500
ARCHT	original	0.000100	1193200	0.435300
ASUS	estimación	0.660283	0.802998	0.481528
BALREP	original	0.000000	1062100	0.029700
BASE	estimación	-0.151175	-0.622104	0.981591
BB	estimación	1446567	0.142237	0.436546
CAT01	original	0.000000	0.784600	0.193000
CAT02	original	0.000000	0.800600	0.470800
CHANK	estimación	1114557	-0.442812	0.581542
CHE	estimación	-0.194803	-0.744389	-1724341

Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Tabla 3

RandomForest

WL	STATE	C/G	G/N	P/G
AGUACLARA	original	0.000000	1068000	0.039800
AGUAL02	original	0.000000	0.820700	0.633500
ARCHT	original	0.000100	1193200	0.435300
ASUS	estimación	-0.354381	0.316765	0.165834
BALREP	original	0.000000	1062100	0.029700
BASE	estimación	-0.756793	0.178514	0.685760
BB	estimación	0.645305	0.325436	0.434394
CAT01	original	0.000000	0.784600	0.193000
CAT02	original	0.000000	0.800600	0.470800
CHANK	estimación	1171950	-0.769068	0.343855
CHE	estimación	0.408885	-1169162	-1520161

Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Los resultados del modelo Random Forest mostraron un rendimiento superior al del modelo de regresión lineal, con un error cuadrático medio más bajo y un coeficiente de determinación mayor, lo que indica una mayor precisión en la clasificación. Esto ayudará a decidir si los modelos más complejos justifican su uso para la clasificación y optimización de los quimiotipos (Tabla 4).

Tabla 4

Resultado de comparación 'RandomForest' con 'LinearRegression'

Modelo	MSE	R^2
Linear Regression	11191	-0.0924
Random Forest	0.6932	0.3110

Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Interpretación de los resultados

Linear Regression

MSE: 1.119: Este valor es relativamente alto, lo que indica que las predicciones de la regresión lineal tienen un error significativo con respecto a los valores reales.

R^2: -0.0924: Un R^2 negativo indica que el modelo no está capturando las relaciones en los datos de manera efectiva. De hecho, un valor negativo de R^2 sugiere que el modelo es peor que un modelo que simplemente predice la media de los valores de salida.

Random Forest

MSE: 0.693: Este valor es más bajo que el de la regresión lineal, lo que sugiere que las predicciones de Random Forest son más precisas.

R^2: 0.311: Un R^2 positivo y más cercano a 1 indica que el modelo de Random Forest es capaz de explicar aproximadamente el 31% de la variabilidad en los datos de salida. Aunque este valor no es extremadamente alto, es significativamente mejor que el R^2 negativo de la regresión lineal.

Mejor Rendimiento de Random Forest: Los resultados sugieren que RandomForestRegressor es capaz de capturar relaciones más complejas y no lineales en los datos de entrada, mientras que LinearRegression falla en capturar estas relaciones. Esto es evidente por el menor MSE y el R^2 positivo de Random Forest en comparación con la regresión lineal.

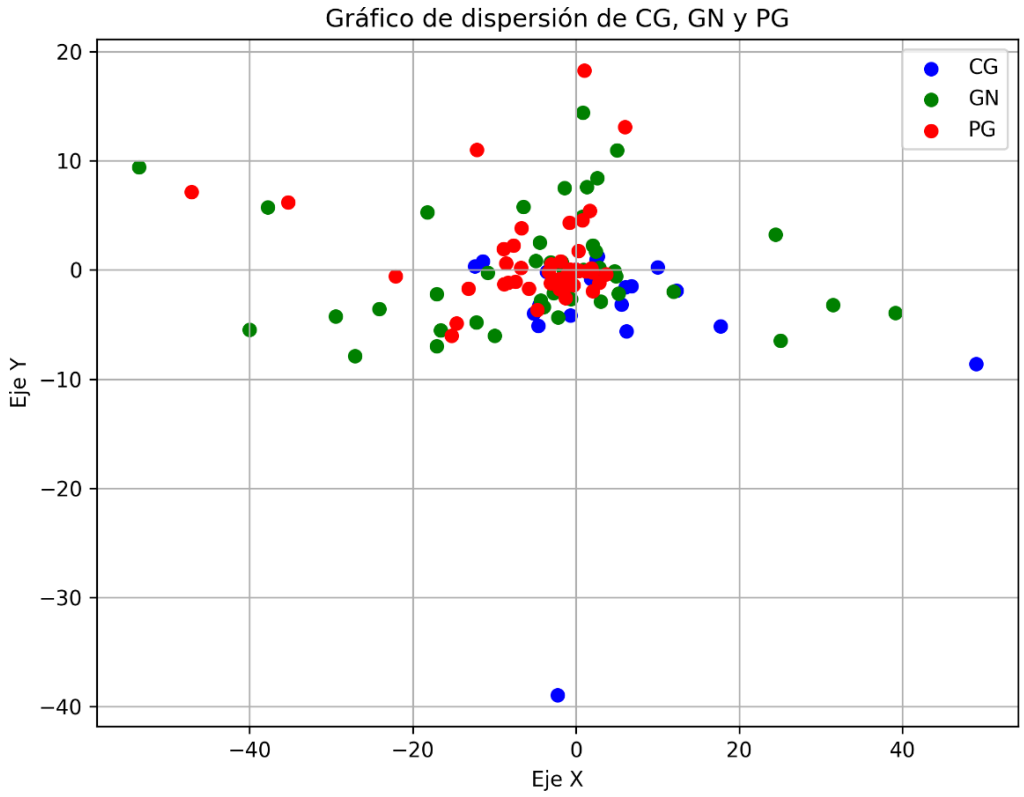
Reducción de Dimensionalidad

El uso de PCA permitió reducir la dimensionalidad de los datos sin perder información relevante, revelando los patrones subyacentes en las muestras de aceites esenciales. Esto facilitó una mejor comprensión de las relaciones entre las variables y optimizó la precisión en la clasificación de los quimiotipos.

Multiplicación del valor de PCA por las columnas C/G. G/N, P/G, esto ayuda a capturar la mayor parte de la varianza presente en los datos originales con menos características lo que conlleva a transformar y ajustar los datos de manera que sean más adecuados para los análisis posteriores (Ilustración 1).

Ilustración 1

Gráfico de dispersión del valor de PCA por las columnas C/G. G/N, P/G



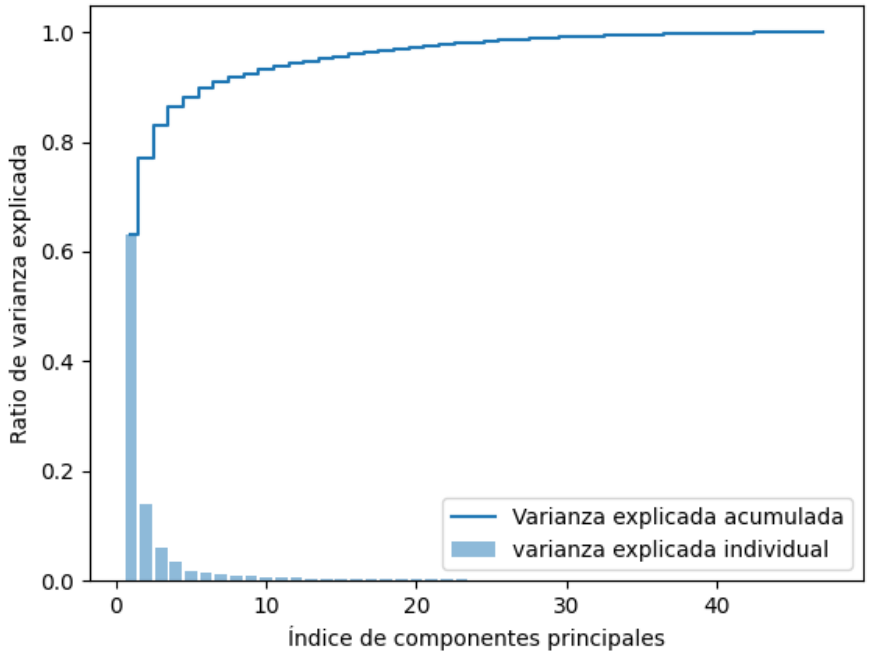
Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Análisis de componentes principales, usando valores y vectores propios de una matriz de covarianza para entender la variabilidad de los datos en un conjunto de entrenamiento estandarizado. El grafico indica cuantos componentes principales se necesitarían para explicar un cierto porcentaje de la variabilidad total de datos (Ilustración 2).

Ilustración 2

Gráfico de varianza explicada por cada componente principal y varianza acumulada



Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

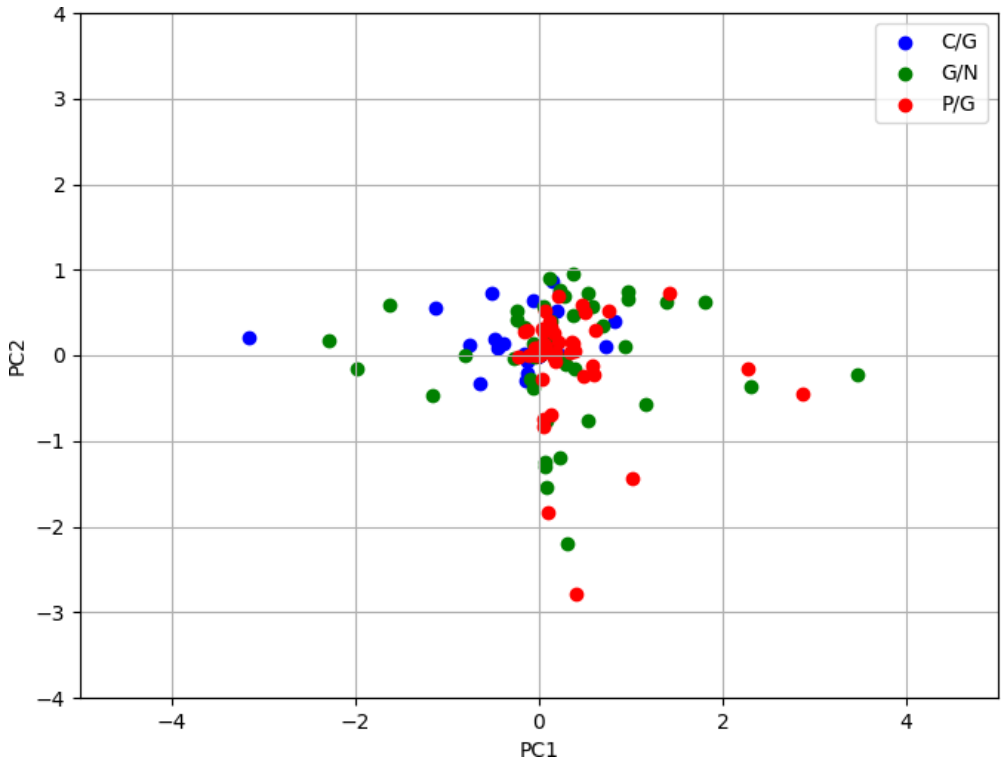
Se forma la matriz W a partir de dos vectores propios principales seleccionados de la lista. Esta matriz se utiliza para proyectar los datos en un espacio de componentes principales reducido. Luego de formar la matriz W se normaliza para tener una media de 0 y una desviación estándar de 1, la cual ayuda a mantener una escala consistente.

Multiplicación por elemento entre la matriz W estandarizada y los valores de las características C/G, G/N y P/G de cada muestra. Esta multiplicación proyecta cada punto en el

nuevo espacio de componentes principales y es esencial para la visualización de los datos en términos de sus componentes principales (Ilustración 3).

Ilustración 3

Gráfico de la multiplicación entre la matriz W estandarizada y los valores de las características C/G, G/N y P/G



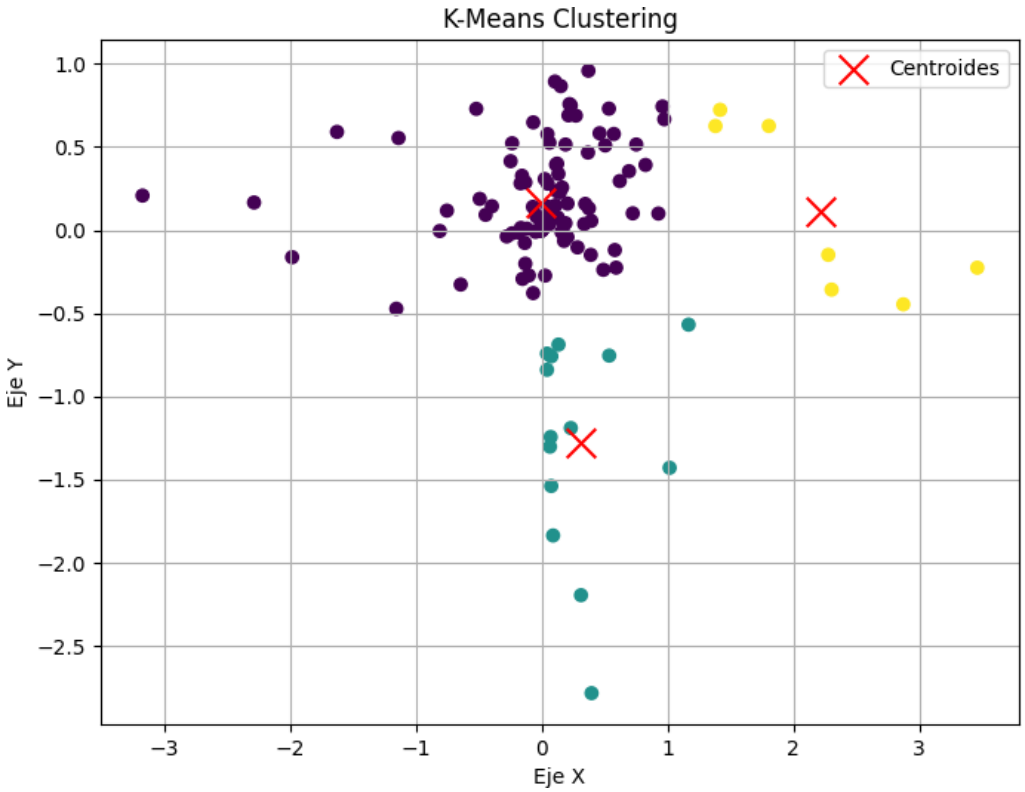
Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Se configura un modelo de K-means para encontrar 3 clústeres en los datos combinados. El algoritmo ejecuta el proceso de inicialización de los centroides 10 veces para encontrar la mejor configuración, lo que ayuda a evitar resultados subóptimos (Ilustración 4).

Ilustración 4

Gráfico de la agrupación de los datos combinados con los tres conjuntos C/G, G/N y P/G



Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

A continuación, se trata con el DataFrame ‘datosEsR’ que contiene valores negativos como resultado de aplicar RandomForest para rellenar valores faltantes (NaN). Los valores negativos no tienen sentido en el contexto de los datos, por lo que es necesario convertirlos a valores absolutos y luego normalizarlos (Tabla 5).

Tabla 5

Se muestra los valores absolutos y luego normalizados

WL	C/G	G/N	P/G
ASUS	0.000000	0.187398	0.066913
BASE	0.214731	0.054294	0.518424
BB	0.125360	0.082317	0.272666
CHANK	0.507802	0.424500	0.136988
CHE	0.054228	1000000	1000000

Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

Se realiza un análisis de agrupamiento (clustering) usando el algoritmo de K-means para identificar patrones o similitudes entre los datos de quimiotipos de aceites esenciales. Se crean gráficos para visualizar cómo se agrupan las muestras en función de las características C/G, G/N y P/G. Cada clúster se representa con un color diferente, y se añaden etiquetas para los nombres de los aceites esenciales para facilitar la interpretación.

Aquí se presentan los puntos de datos agrupados en seis clústeres diferentes, cada uno representado con un color distinto. Los nombres de los aceites esenciales están etiquetados junto a sus respectivos puntos, lo que facilita la identificación de la pertenencia a un clúster específico (Ilustración 5).

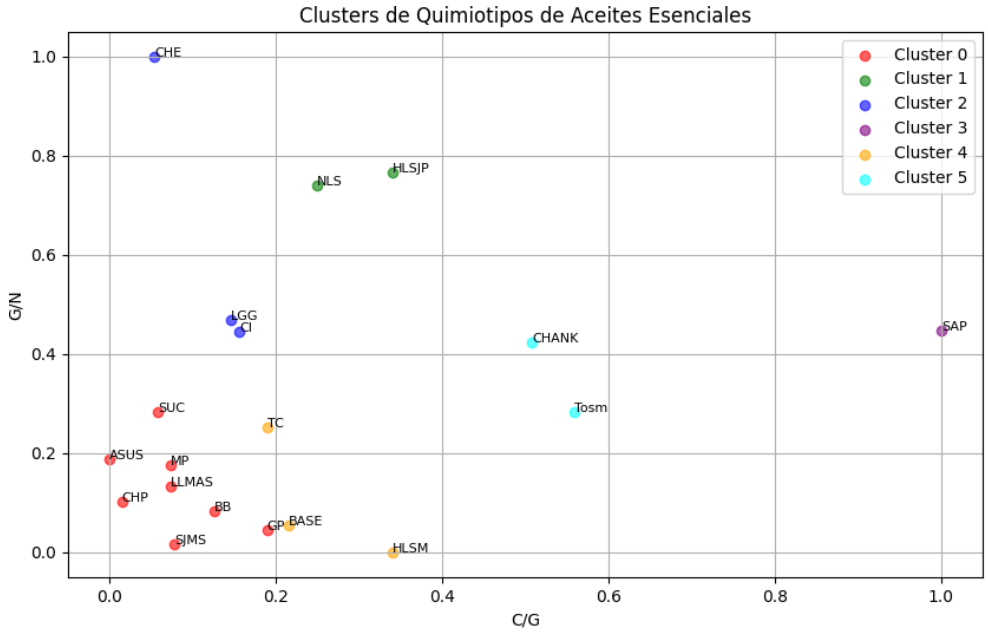
Distribución de Clústeres

Los aceites esenciales están agrupados en seis clústeres, identificados por colores:

- **Clúster 0 (Rojo)**: Contiene aceites con valores relativamente bajos de C/G y G/N. Este clúster parece representar una categoría de aceites esenciales con características químicas similares, que pueden ser considerados como una base de comparación con otros clústeres.
- **Clúster 1 (Verde)**: Incluye aceites esenciales como HLSJP y NLS, que tienen valores moderadamente altos de G/N y bajos de C/G. Este grupo sugiere una distinción química en comparación con los otros clústeres.
- **Clúster 2 (Azul)**: Este clúster tiene una alta proporción de G/N, destacando aceites esenciales como CHE y LGG, que podrían tener propiedades específicas relacionadas con su composición química.
- **Clúster 3 (Morado)**: Contiene sólo el aceite SAP, el cual se distingue por su alto valor en C/G. Esto indica una composición única que podría ser significativa para aplicaciones específicas.
- **Clúster 4 (Naranja)**: Agrupa aceites como BASE, GP, y LLMAS, con valores moderados de C/G y bajos en G/N. Este clúster podría representar una subcategoría de aceites con características menos pronunciadas en términos de C/G.
- **Clúster 5 (Cian)**: Contiene aceites como CHANK y Tosm, ubicados en regiones con valores intermedios de C/G y G/N. Este grupo puede indicar una categoría de aceites con un balance de estas dos características.

Ilustración 5

Resultados del análisis de clustering aplicado a los quimiotipos de aceites esenciales



Fuente: Elaboración propia

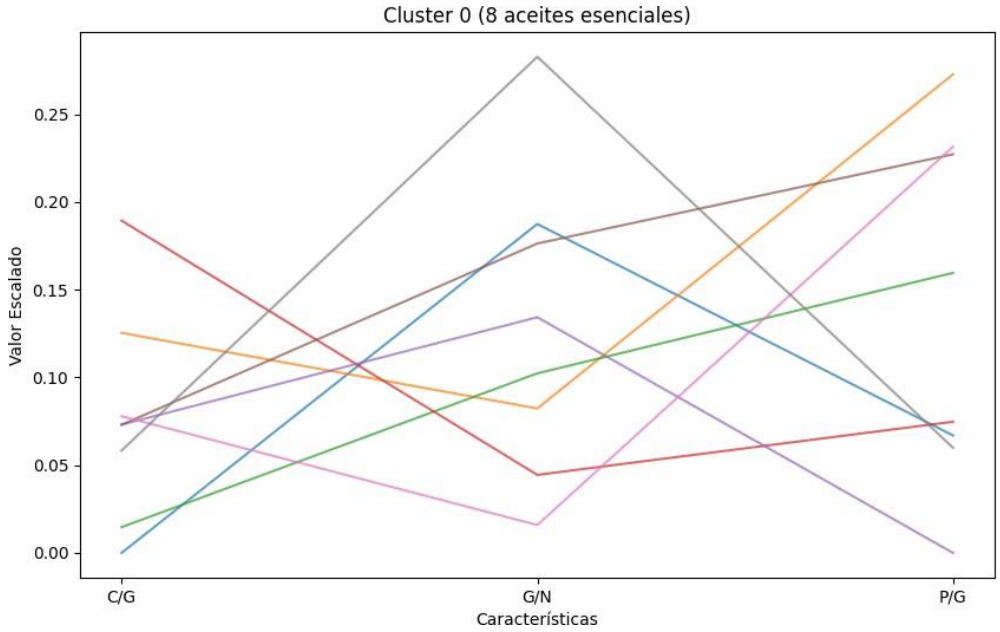
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 0 (8 aceites esenciales)

Este grupo tiene valores bajos de ‘C/G’ y ‘G/N’, pero muestra una variedad en ‘P/G’ (Pineo/Geranial). Esto puede sugerir que estos aceites esenciales tienen una menor concentración de compuestos específicos, pero todavía se agrupan debido a otras características comunes (Ilustración 6).

Ilustración 6

Clúster 0



Fuente: Elaboración propia

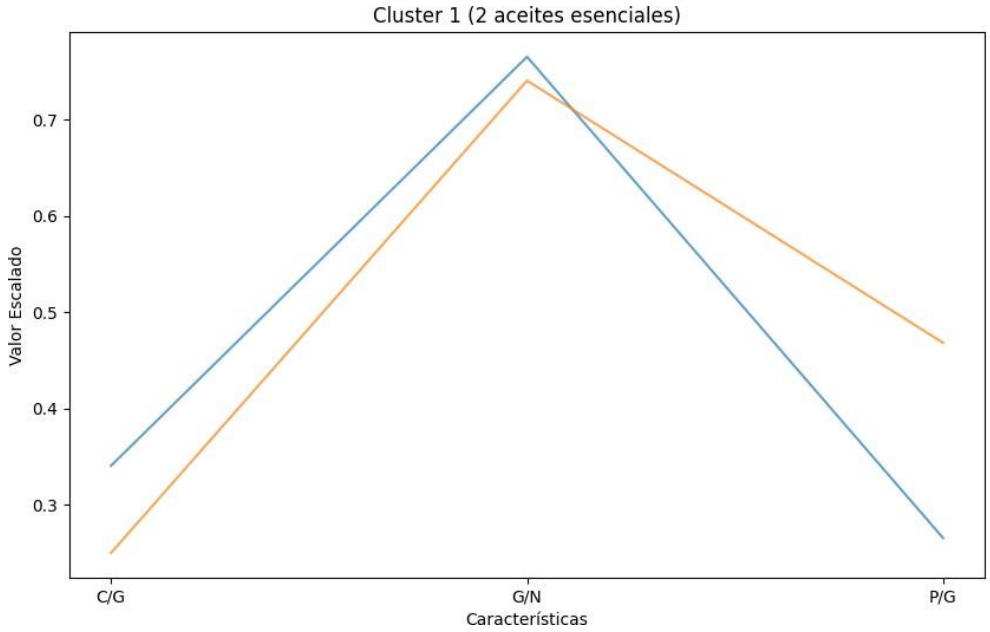
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 1 (2 aceites esenciales)

Estos aceites muestran valores altos de G/N, lo que sugiere una alta proporción de Geranial en comparación con Neral. Esto puede indicar que estos aceites tienen un perfil aromático más fuerte asociado con el Geranial (Ilustración 7).

Ilustración 7

Clúster 1



Fuente: Elaboración propia

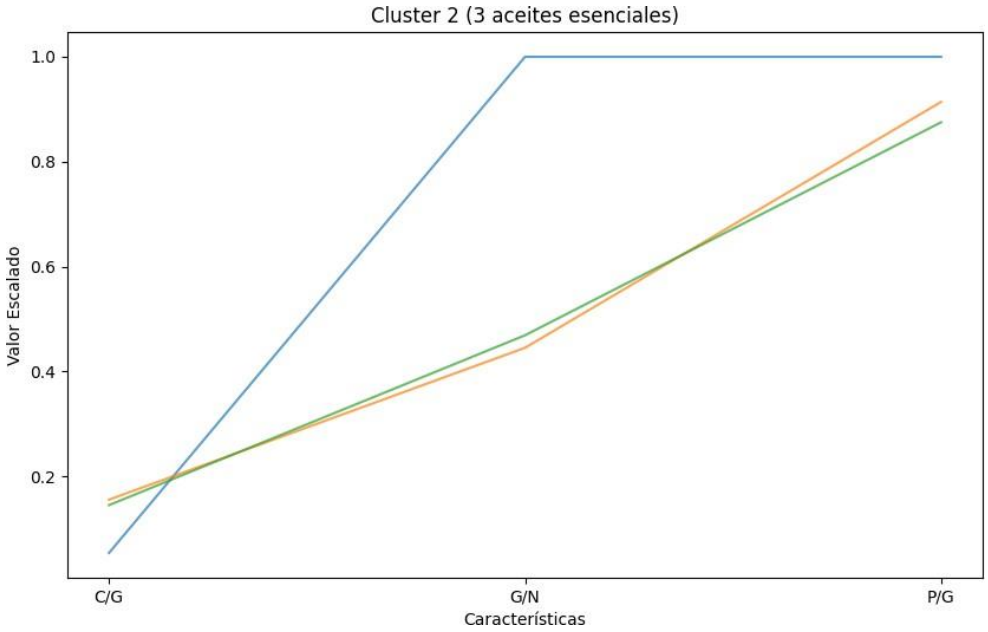
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 2 (3 aceites esenciales)

Los aceites en este clúster tienen valores altos de C/G, indicando una alta proporción de Careno en comparación con Geranial, pero se equilibran con valores moderados de G/N y P/G (Ilustración 8).

Ilustración 8

Clúster 2



Fuente: Elaboración propia

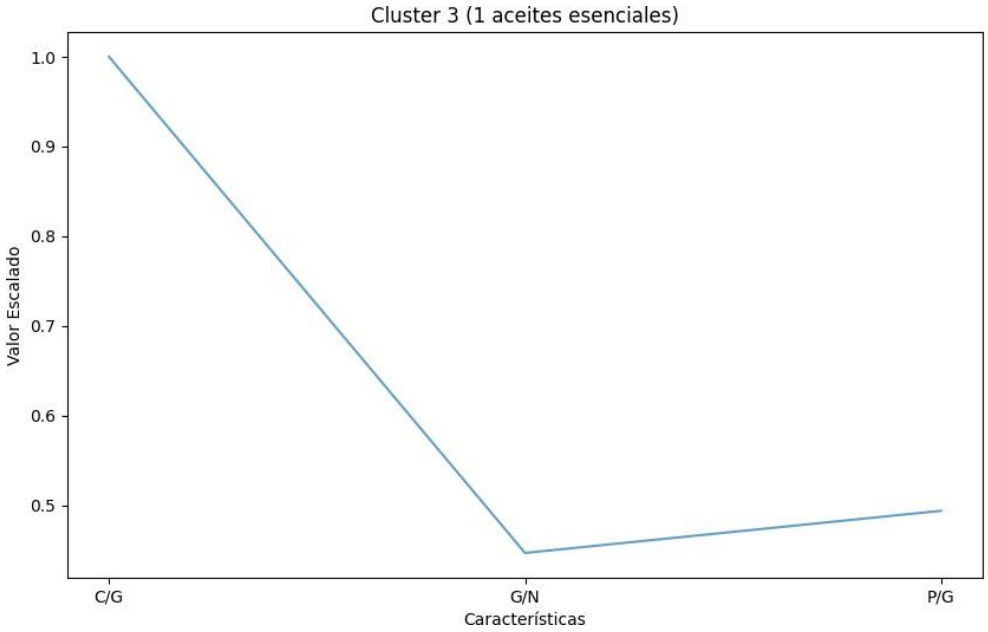
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 3 (1 aceite esencial)

Solo un aceite esencial cae en este clúster, mostrando un valor extremo en C/G y un bajo valor en G/N, lo que lo hace único en su composición (Ilustración 9).

Ilustración 9

Clúster 3



Fuente: Elaboración propia

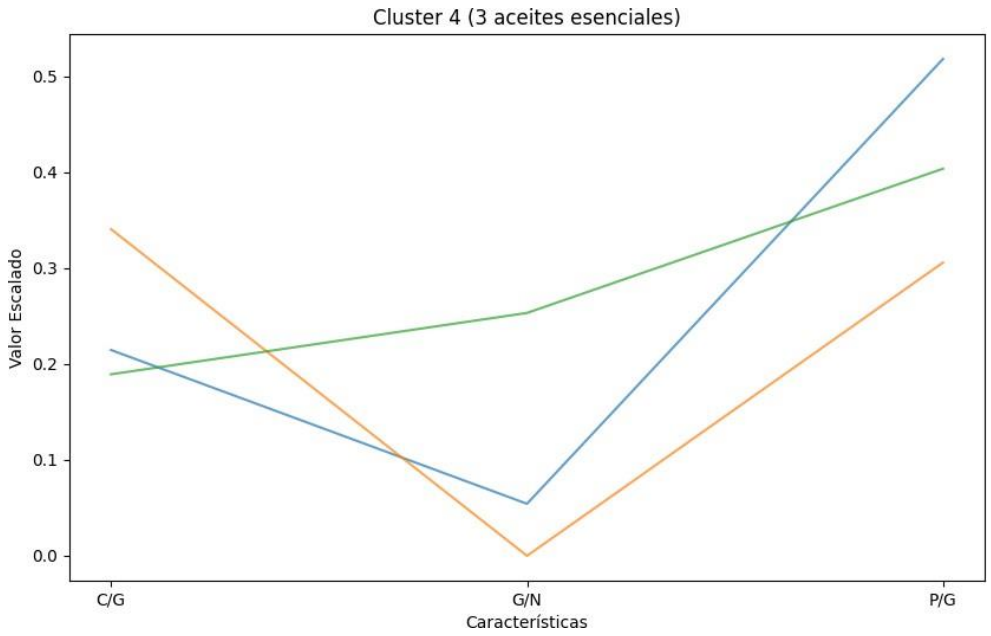
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 4 (3 aceites esenciales)

Estos aceites tienen valores moderados en todas las características, lo que indica una mezcla balanceada de los componentes principales (ilustración 10).

Ilustración 10

Clúster 4



Fuente: Elaboración propia

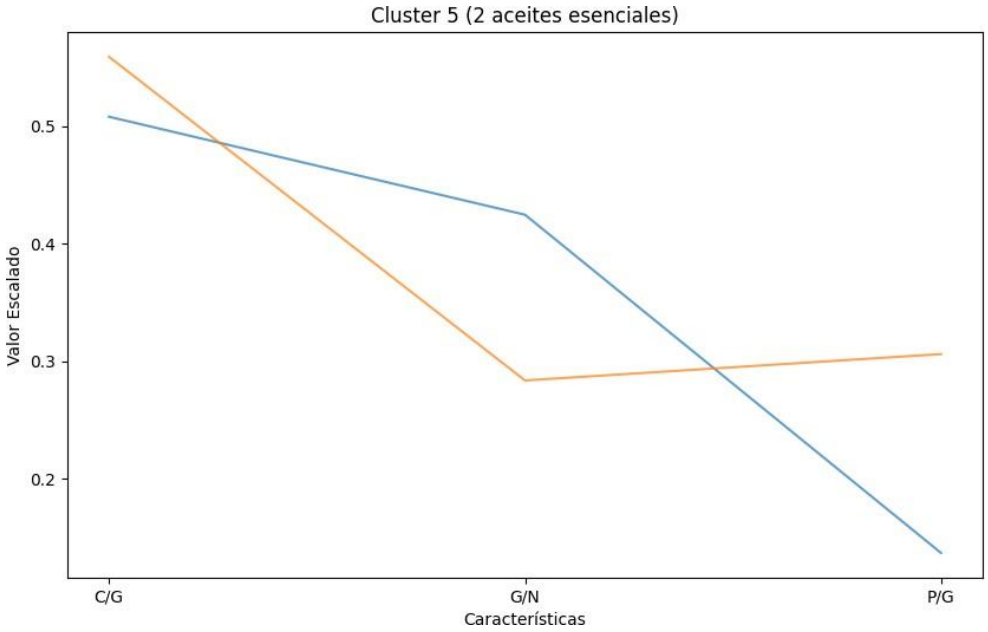
Autor: Cereceda Guzmán Borys Álvaro.

Clúster 5 (2 aceites esenciales)

Este grupo muestra una disminución progresiva en los valores de C/G, G/N y P/G, lo que sugiere que estos aceites son más consistentes en su perfil químico en comparación con otros clústeres (ilustración 11).

Ilustración 11

Clúster 5



Fuente: Elaboración propia

Autor: Cereceda Guzmán Borys Álvaro.

CONCLUSIONES

El uso de Machine Learning en la clasificación de quimiotipos de aceites esenciales de Cymbopogon citratus demostró ser una herramienta eficaz, superando a los métodos tradicionales como la regresión lineal en la precisión de predicción y optimización de los compuestos químicos.

El algoritmo Random Forest mostró un mejor desempeño que el modelo de regresión lineal, con un menor error cuadrático medio (MSE) y un coeficiente de determinación positivo, lo que sugiere que puede capturar mejor las relaciones no lineales presentes en los datos espectrales.

La reducción de dimensionalidad mediante PCA permitió simplificar el conjunto de datos sin perder información relevante, lo que facilitó la visualización de los patrones subyacentes en las muestras de aceites esenciales.

REFERENCIAS BIBLIOGRÁFICAS

A. Alam. (2023). What is Machine Learning? https://www.researchgate.net/publication/373015635_What_is_Machine_Learning

Aldi, FH (2023, 10 de diciembre). Potencial de StandardScaler para mejorar la precisión en el cáncer de mama mediante el aprendizaje automático. Revista de Ingeniería Aplicada y Ciencia Tecnológica, 5 (1). https://doi.org/10.37385/jaets.v5i1.3080

Al-Hadi, AH y Hasnu, AHI (2022, 17 de septiembre). Estudio preliminar sobre el aceite esencial de madera de agar y sus técnicas de clasificación mediante aprendizaje automático. Revista Indonesia de Ingeniería Eléctrica e Informática, 29 (2), 753-760. https://doi.org/10.11591/ijeecs.v29.i2.pp753-760

Brown, S. (2021). Machine learning, explained. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Corral, F. (2023, 13 de marzo). Composición de los aceites esenciales: ¿Qué es el quimiotipo? Jabonciudad. https://www.jaboncity.com/post/composici%C3%B3n-de-los-aceites-esenciales-qu%C3%A9-es-el-quimiotipo

Gómez Araujo, M. E. (octubre de 2017). Análisis multivariado de componentes terpénicos, en aceite esencial de hierba luisa (Cymbopogon citratus), mediante espectrofotometría UV - Visible Derivada. http://dspace.ups.edu.ec/handle/123456789/14765

Gurrea, MT (sf). Análisis de Componentes Principales. Universidad de Santiago de Compostela. http://eio.usc.es/pub/mjginzo/descargas/leyenda/Documents/seleccion%20de%20variables/776B41F8d01.pdf

Hartigan, JA (1979). Un algoritmo de agrupamiento de K-medias. Revista de la Royal Statistical Society: Serie C (Estadística Aplicada), 28 (1), 100-108. https://doi.org/10.2307/2346830

H2O.ai. (2024). ¿Qué es el aprendizaje automático supervisado y cómo se utiliza? Documentación de H2O.ai. https://h2o.ai

Hinton, GE y Salakhutdinov, RR (2006). Reducción de la dimensionalidad de los datos con redes neuronales. Science, 313 (5786), 504-507. https://doi.org/10.1126/science.1127647

IBM. (2021). ¿Qué es el aprendizaje supervisado? Centro de aprendizaje de IBM Cloud. https://www.ibm.com/cloud/learn/supervised-learning

IH, S. (22 de marzo de 2021). Aprendizaje automático: algoritmos, aplicaciones reales y direcciones de investigación. doi:10.1007/s42979-021-00592-x

Incorporado. (2024). Aprendizaje supervisado con Python: qué hay que saber. Incorporado. https://builtin.com

Jordan MI, M. T. (2015). Aprendizaje automático: tendencias, perspectivas y perspectivas. doi:10.1126/ciencia.aaa8415.

Kimura, H. H. (Agosto de 2020). Imputación de datos faltantes utilizando datos generados por GAN. doi:https://doi.org/10.1145/3418688.3418701.

Lebanov L, P. B. (2021). Espectómetro Raman portátil basado en teléfono inteligente y aprendizaje automático para la evaluación de la calidad de los aceites esenciales. doi:10.1039/d1ay00886b.

MathWorks, Inc. (s.f.). Linear Regression. https://la.mathworks.com/discovery/linear-regression.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... y Duchesnay, E. (2011). Scikit-learn: aprendizaje automático en Python. Revista de investigación en aprendizaje automático, 12, 2825-2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

S, v. B. (2007). Imputación múltiple de datos discretos y continuos mediante especificación totalmente condicional. doi:10.1177/0962280206074463.

Sarker, IH, Furhad, M. y Nowrozy, R. (2021, 21 de enero). Ciberseguridad impulsada por IA: Descripción general, modelado de inteligencia de seguridad y direcciones de investigación. SN Informática, 2 (3), 1-18. https://doi.org/10.1007/s42979-021-00557-0

Sharifi-Rad, J., Salehi, B., Stojanovic-Radic, Z., Fokou, PVT, Sharifi-Rad, M., Mahady, GB, ... y Martins, N. (2017). Actividades biológicas de los aceites esenciales: De la quimioecología vegetal a los sistemas de curación tradicionales. Moléculas, 22 (1), 70. https://doi.org/10.3390/molecules22010070

Surono, S. (2023). Desarrollo de un modelo optimizado de red neuronal recurrente para la predicción de la calidad del aire utilizando agrupamiento de K-means y reducción de dimensión de PCA. Revista Internacional de Innovación e Investigación en Ciencias de la Educación, 6 (2), 1427. https://doi.org/10.53894/ijirss.v6i2.1427

Tolles, J. y Meurer, WJ (2016). Regresión logística: Relación entre las características de los pacientes y los resultados. JAMA, 316 (10), 1161-1162. https://doi.org/10.1001/jama.2016.7653

Tri Pamungkas, FI (2023). Predicción de ganancias utilizando múltiples métodos de regresión lineal en lenguaje de programación Python en PT Tri Erndov Rezeki. Revista Internacional de Investigación en Economía, Negocios y Contabilidad, 3 (5), 1076. https://doi.org/10.54443/ijebas.v3i5.1076

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., ... y Botstein, D. (2001). Métodos de estimación de valores faltantes para microarrays de ADN. Bioinformática, 17 (6), 520-525. https://doi.org/10.1093/bioinformatics/17.6.520

Ullah, Z., Khattak, MD, Mehmood, A. y Khan, RA (2021, septiembre). Un estudio comparativo de métodos de aprendizaje automático para la predicción del rendimiento de bioaceite:

una selección de características basada en un algoritmo genético. Tecnología de biorrecursos, 325, 125292. https://doi.org/10.1016/j.biortech.2021.125292

UNIR. (07 de julio de 2021). El algoritmo gradient descent para el entrenamiento de redes neuronales. https://www.unir.net/ingenieria/revista/gradientdescent/#:~:text=El%20algoritmo%20de%20gradiente%20descendente,sencillez%20y%20facilidad%20de%20implementaci%C3%B3n.

Viciano-Tudela, S., Peinado, L., Cañete, M., Berna, A., & Ferrero, R. (2023, 22 de junio). Propuesta de un nuevo sistema de clasificación de aceites esenciales basado en sensores de gases de bajo costo y técnicas de aprendizaje automático. Sensores, 23 (13), 5812. https://doi.org/10.3390/s23135812

Xinzhe Zhu, Y. L. (2019). Predicción mediante aprendizaje automático del rendimiento del biocarbón y del contenido de carbono en el biocarbón en función de las características de la biomasa y las condiciones de pirólisis. doi:10.1016/j.biortech.2019.121527

Z, G. (2015). Aprendizaje automático probabilístico e inteligencia artificial. doi:10.1038/nature14541.

Zhang, W. (2023). Comparación de regresión lineal, regresor de árbol de decisión y regresor de bosque aleatorio basado en Python, una empresa de restaurantes en Kaggle como caso. doi:https://doi.org/10.54691/bcpbm.v36i.3449

ANEXOS

Anexo 1.

DataFrame

WL	C/G	G/N	P/G	400	399.5	399
AGUACLARA	0	1068	0.0398	0.0389111	0.0405307	0.0420104
AGUAL02	0	0.8207	0.6335	0.0912953	0.094527	0.0975215
ARCHT	0.0001	11932	0.4353	0.0556569	0.0582357	0.0606056
ASUS				0.0598808	0.0627314	0.0653313
BALREP	0	10621	0.0297	0.0574284	0.05992	0.0621791
BASE				0.0657298	0.0689938	0.0720074
BB				0.0593927	0.062003	0.0643823
CAT01	0	0.7846	0.193	0.0514417	0.0541702	0.0566385
CAT02	0	0.8006	0.4708	0.0569595	0.0597718	0.0623229
CHANK				0.0599329	0.0625638	0.0649601
CHE				0.0296826	0.0309597	0.0321279
CHP				0.0594007	0.0621725	0.0646972
CI				0.0841666	0.0877418	0.0910527
CTRON01	0	0.8397	0.5696	0.0436475	0.0456925	0.047542
CTRON02	0	0.8368	0.4346	0.0594245	0.0623073	0.0649261
GP				0.0462491	0.0485225	0.050603
HLSJP				0.0339435	0.0353824	0.0366679
HLSM				0.0674518	0.0708136	0.0739044
JIM	0	0.9308	0.8197	0.0563681	0.0585565	0.0605198

Nota. Adaptada de Análisis multivariado de componentes terpénicos en aceite esencial de Hierbaluisa (Cymbopogon citratus), mediante espectrofotometría UV - visible derivada. Por G. Araujo, T. Gonzaga, 2017.

La tabla muestra una selección representativa de los datos iniciales utilizados en este estudio, provenientes del archivo data.csv. Cada fila de la tabla corresponde a una muestra específica de aceite esencial de *Cymbopogon citratus* (Hierbaluisa), identificada por un nombre de muestra. Las columnas incluyen ratios de compuestos químicos, como Careno/Geranial (C/G), Geranial/Neral (G/N), y Pineo/Geranial (P/G), así como valores de absorbancia espectral medidos en diferentes longitudes de onda (por ejemplo, 400 nm, 399.5 nm, 398 nm). Estos datos fueron

utilizados para aplicar técnicas de Machine Learning con el fin de clasificar y optimizar los quimiotipos de los aceites esenciales.

Anexo 2.

Código en Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
import os

from google.colab import drive
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler



"""
    Estandariza los datos al restar la media y dividir por la desviación
estándar.
    Args:
    - ex (pd.DataFrame): Datos a estandarizar.

    Returns:
    - pd.DataFrame: Datos estandarizados.
    """

def estandarizar(ex):
    return (ex - ex.mean()) / ex.std()



drive.mount('/content/drive')
np.set_printoptions(precision=10, suppress=True)
datos = pd.read_csv("/content/drive/My Drive/data.csv")
print(datos.head())
datos.shape
total_nan =datos.isnull().any(axis=1)
totalF = total_nan.sum()

nan_rows = datos[pd.isnull(datos).any(axis=1)]
print("Filas con NaN:\n", nan_rows)
```

```
completos =datos.dropna()
t_comp =datos.iloc[:, 2:5].copy()
t_esp = datos.iloc[:, 5:].copy()

t_espe_std = estandarizar(datos.iloc[:, 5:].copy())
c_comp_std = estandarizar(completos.iloc[:, 2:5].copy())


c_espe_std = t_espe_std[t_espe_std.index.isin(c_comp_std.index)].copy()

from sklearn.linear_model import LogisticRegression

X= c_espe_std
y= c_comp_std

X,y

from sklearn.impute import SimpleImputer

datos_numericos = datos.select_dtypes(include='number')

imputador = SimpleImputer(strategy='mean')

imputador.fit(datos_numericos)

datos_imputados_numericos = imputador.transform(datos_numericos)

datos_imputados_numericos = pd.DataFrame(datos_imputados_numericos,
columns=datos_numericos.columns)

datos_imputados = pd.concat([datos['WL'], datos_imputados_numericos], axis=1)

print(datos_imputados)

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import os
import matplotlib.pyplot as plt
 """
    Clase para la implementación de un modelo lineal.
    Args:
    - nombre (str): Nombre del modelo.
    - model (object): Modelo a utilizar (por ejemplo, LinearRegression).
    """
class ModelW:

    def __init__(self, nombre,
        model):self.nombre = nombre
        self.model = model
        self.estimaciones = None
        self.combinado = None
```

```
"""
        Ajusta el modelo con los datos proporcionados.
        Args:
        - X (array-like): Datos de entrada.
        - y (array-like): Datos de salida.
        """
    def fit(self, X, y):
        self.model.fit(X, y)


 """
        Realiza predicciones con los datos proporcionados.
        Args:
        - X (array-like): Datos de entrada.

        Returns:
        - array: Predicciones del modelo.
        """
    def predict(self, X):


        return self.model.predict(X)

 """
        Ejecuta el modelo, predice y combina los resultados con los datos
originales.

        Returns:
        - pd.DataFrame: Datos combinados con las estimaciones.
        """
    def ejecutar(self):

        if not os.path.exists(f'output/{self.nombre}'):
            os.makedirs(f'output/{self.nombre}')

        self.estimaciones = pd.DataFrame(columns=["est_C/G", "est_G/N",
"est_P/G"],
                                         data=self.predict(t_espe_std))
        self.estimaciones.insert(loc=0, column="WL",
value=datos_imputados["WL"])

        self.combinado = datos.copy()
        self.indices_vacios =
self.combinado[self.combinado["C/G"].isna()].index
        self.combinado['STATE'] = 'original'
        self.combinado.loc[self.combinado["C/G"].isna(), 'STATE'] =
'estimacion'

        # Convertir las predicciones a float explícitamente para evitar
warnings
        self.combinado.loc[self.combinado["C/G"].isna(), 'C/G'] =
self.estimaciones.values[:, 1][self.indices_vacios].astype(float)
```

```
        self.combinado.loc[self.combinado["G/N"].isna(), 'G/N'] =
self.estimaciones.values[:, 2][self.indices_vacios].astype(float)
        self.combinado.loc[self.combinado["P/G"].isna(), 'P/G'] =
self.estimaciones.values[:, 3][self.indices_vacios].astype(float)

        self.combinado = self.combinado[["WL", "STATE", "C/G", "G/N", "P/G"]]
        print(self.combinado)
        return self.combinado
"""
        Guarda los resultados de las estimaciones y datos combinados en
archivos CSV.
        """
    def guardar(self):

        if not os.path.exists('/content/drive/My Drive/estimaciones'):
            os.makedirs('/content/drive/My Drive/estimaciones',
exist_ok=True)
        if not os.path.exists('/content/drive/My Drive/combinado'):
            os.makedirs('/content/drive/My Drive/combinado')

        self.estimaciones.to_csv(f"/content/drive/My
Drive/estimaciones/{self.nombre}_estimaciones.csv", index=False)
        self.combinado.to_csv(f"/content/drive/My
Drive/combinado/{self.nombre}_combinado.csv", index=False)
 """
        Retorna los datos originales (sin estimaciones).

        Returns:
        - pd.DataFrame: Filas que contienen los datos originales.
        """
    def get_original_data(self):

        return self.combinado[self.combinado["STATE"] == 'original']
"""
        Retorna las filas con datos estimados.

        Returns:
        - pd.DataFrame: Filas que contienen estimaciones.
        """
    def get_estimacion_data(self):

        return self.combinado[self.combinado["STATE"] == 'estimacion']
"""
        Genera un gráfico de dispersión de los datos originales y las
estimaciones.
        """
    def graficar(self):

        original_data = self.combinado[self.combinado["STATE"] == 'original']
        estimacion_data = self.combinado[self.combinado["STATE"] ==
'estimacion']

        print("Datos para la grafica 'original': ")
```

```
        print(original_data.head())
        print("Datos para la grafica 'estimacion': ")
        print(estimacion_data.head())

        plt.scatter(estimacion_data['G/N'], estimacion_data["P/G"], c='blue',
label='Original')
        plt.scatter(estimacion_data['C/G'], estimacion_data["P/G"], c='red',
label='Estimado')
        plt.scatter(estimacion_data['G/N'], estimacion_data["C/G"],
c='green', label='C/G')
        plt.xlabel('G/N')
        plt.ylabel('P/G / C/G')
        plt.legend()
        plt.show()

modelos = {
    "linear": LinearRegression()
}

for m in modelos:
  modelo = ModelW(m, modelos [m])
  modelo.fit(X, y.values)
  modelo.ejecutar()
  modelo.guardar()

  error=(modelo.combinado.values[c_comp_std.index][:, 3:6]-
modelo.estimaciones.values[c_comp_std.index][:, 2:5]).astype(float)
  print(error)

  modelo.graficar()


  import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
import os
import matplotlib.pyplot as plt
 """
    Clase para implementar y gestionar un modelo basado en Random Forest.

    Args:
    - nombre (str): Nombre del modelo.
    - model (object): Modelo de scikit-learn, por ejemplo,
RandomForestRegressor.
    """
class ModelRandomForest:

    def__init__(self, nombre,
        model):self.nombre = nombre
        self.model = model
        self.estimaciones = None
        self.combinado = None
```

```
    def fit(self, X, y):
        self.model.fit(X, y)

    def predict(self, X):
        return self.model.predict(X)

    def ejecutar(self):
        if not os.path.exists(f'output/{self.nombre}'):
            os.makedirs(f'output/{self.nombre}')

        self.estimaciones = pd.DataFrame(columns=["est_C/G", "est_G/N",
"est_P/G"],
                                         data=self.predict(t_espe_std))
        self.estimaciones.insert(loc=0, column="WL",
value=datos_imputados["WL"])

        self.combinado = datos.copy()
        self.indices_vacios =
self.combinado[self.combinado["C/G"].isna()].index
        self.combinado['STATE'] = 'original'
        self.combinado.loc[self.combinado["C/G"].isna(), 'STATE'] =
'estimacion'

        # Convertir las predicciones a float explícitamente para evitar
warnings
        self.combinado.loc[self.combinado["C/G"].isna(), 'C/G'] =
self.estimaciones.values[:, 1][self.indices_vacios].astype(float)
        self.combinado.loc[self.combinado["G/N"].isna(), 'G/N'] =
self.estimaciones.values[:, 2][self.indices_vacios].astype(float)
        self.combinado.loc[self.combinado["P/G"].isna(), 'P/G'] =
self.estimaciones.values[:, 3][self.indices_vacios].astype(float)

        self.combinado = self.combinado[["WL", "STATE", "C/G", "G/N", "P/G"]]
        print(self.combinado)
        return self.combinado

    def guardar(self):
        if not os.path.exists('/content/drive/My Drive/estimaciones'):
            os.makedirs('/content/drive/My Drive/estimaciones',
exist_ok=True)
        if not os.path.exists('/content/drive/My Drive/combinado'):
            os.makedirs('/content/drive/My Drive/combinado')

        self.estimaciones.to_csv(f"/content/drive/My
Drive/estimaciones/{self.nombre}_estimaciones.csv", index=False)
        self.combinado.to_csv(f"/content/drive/My
Drive/combinado/{self.nombre}_combinado.csv", index=False)

    def get_original_data(self):
        return self.combinado[self.combinado["STATE"] == 'original']

    def get_estimacion_data(self):
        return self.combinado[self.combinado["STATE"] == 'estimacion']
```

```
    def graficar(self):
        original_data = self.combinado[self.combinado["STATE"] == 'original']
        estimacion_data = self.combinado[self.combinado["STATE"] ==
'estimacion']

        print("Datos para la grafica 'original': ")
        print(original_data.head())
        print("Datos para la grafica 'estimacion': ")
        print(estimacion_data.head())

        plt.scatter(estimacion_data['G/N'], estimacion_data["P/G"], c='blue',
label='Original')
        plt.scatter(estimacion_data['C/G'], estimacion_data["P/G"], c='red',
label='Estimado')
        plt.scatter(estimacion_data['G/N'], estimacion_data["C/G"],
c='green', label='C/G')
        plt.xlabel('G/N')
        plt.ylabel('P/G / C/G')
        plt.legend()
        plt.show()

# Definir el modelo RandomForest y crear la instancia de la nueva clase
modelos_rf = {
    "random_forest": RandomForestRegressor(n_estimators=100, random_state=42)
}

for m in modelos_rf:
    modelo_rf = ModelRandomForest(m, modelos_rf[m])
    modelo_rf.fit(X, y.values)
    modelo_rf.ejecutar()
    modelo_rf.guardar()

    error_rf = (modelo_rf.combinado.values[c_comp_std.index][:, 3:6] -
                modelo_rf.estimaciones.values[c_comp_std.index][:,
                                              2:5]).astype(float)
    print(error_rf)

    modelo_rf.graficar()

# Crear una instancia de la clase ModelW con LinearRegression
modelo = ModelW("linear", LinearRegression())

# Ajustar el modelo con los datos de entrada
modelo.fit(X, y.values)

# Ejecutar el modelo para generar estimaciones y almacenar los resultados
datos_generados = modelo.ejecutar()  # Llamada única a ejecutar()

# Imprimir los datos generados
print("DATOS GENERADOS POR LINEAR REGRESSION")
print(datos_generados)
```

```
modeloR = ModelRandomForest("random_forest", RandomForestRegressor())
modeloR.fit(X, y.values)

# Ejecutar el modelo para generar estimaciones y almacenar los resultados
datos_generadosR = modeloR.ejecutar()  # Llamada única a ejecutar()

# Imprimir los datos generados
print("DATOS GENERADOS POR RANDOM FOREST")
print(datos_generadosR)

import pandas as pd



# Eliminar la columna 'STATE'
datos_sin_state = datos_generados.drop(columns=['STATE'])

# Filtrar solo las filas donde 'STATE' es 'estimacion'
datos_estimacion = datos_generados[datos_generados['STATE'] == 'estimacion']

# Eliminar la columna 'STATE' en el DataFrame filtrado
datosEs = datos_estimacion.drop(columns=['STATE'])

# Mostrar el resultado
print("RESULTADO LINEAR REGRESSION")
print(datosEs)

# Eliminar la columna 'STATE'
datos_sin_state = datos_generadosR.drop(columns=['STATE'])

# Filtrar solo las filas donde 'STATE' es 'estimacion'
datos_estimacion = datos_generadosR[datos_generadosR['STATE'] ==
'estimacion']

# Eliminar la columna 'STATE' en el DataFrame filtrado
datosEsR = datos_estimacion.drop(columns=['STATE'])

# Mostrar el resultado
print("RESULTADO LINEAR RANDOM FOREST")
print(datosEsR)


import pandas as pd
data_numeric = datos_generados.drop(columns=['WL', 'STATE'])
data_numeric[['C/G', 'G/N', 'P/G']] = data_numeric[['C/G', 'G/N',
'P/G']].astype(float)
data_numeric = data_numeric[(data_numeric[['C/G', 'G/N', 'P/G']] !=
0).any(axis=1)]

print(data_numeric)

import pandas as pd
data_numericR = datos_generadosR.drop(columns=['WL', 'STATE'])
```

```
data_numericR[['C/G', 'G/N', 'P/G']] = data_numericR[['C/G', 'G/N',
'P/G']].astype(float)
data_numericR = data_numericR[(data_numericR[['C/G', 'G/N', 'P/G']] !=
0).any(axis=1)]

print(data_numericR)


from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

X = c_espe_std
y = c_comp_std

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Entrenar el modelo de Regresión Lineal
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

# Evaluar el rendimiento del modelo de Regresión Lineal
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)
print(f"Linear Regression - MSE: {mse_linear}, R^2: {r2_linear}")

# Entrenar el modelo de Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Evaluar el rendimiento del modelo de Random Forest
mse_rf = mean_squared_error(y_test,  y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest - MSE: {mse_rf}, R^2: {r2_rf}")

import pandas as pd
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()
normalized_data = normalizer.fit_transform(data_numericR)
normalized_df = pd.DataFrame(normalized_data, columns=['C/G', 'G/N', 'P/G'])

print("DataFrame con datos normalizados:")
print(normalized_df)

from sklearn.model_selection import train_test_split

X, y = datos_imputados.iloc[:, 1:].values, datos_imputados.iloc[:, 0].values
```

```
X_train, y_train = X, y
X_test, y_test = None, None

# Comprobar las formas de X_train y X_test
print("Forma de X_train:", X_train.shape)


from sklearn.preprocessing import Normalizer

data = X_train
normalizer = Normalizer()

normalized_X_train = normalizer.fit_transform(data)
print("\nDatos normalizados:")
print(normalized_X_train)
normalized_data.shape
#Estandarizamos
scaler = StandardScaler()
X_train_std = scaler.fit_transform(normalized_X_train)

#PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)

import numpy as np

A = X_train_pca
B = datos_generadosR['C/G'].values.reshape(-1, 1)

R_cg = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_cg[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_cg)
print("Forma de R:", R_cg.shape)

plt.scatter(R_cg[:, 0], R_cg[:, 1], c='red')
plt.title('Grafico de dispersion C/G')
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
plt.show()
plt.savefig("grafica1.eps", format='eps')
plt.close()

import numpy as np

A = X_train_pca
B = datos_generadosR['G/N'].values.reshape(-1, 1)
```

```
R_gn = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_gn[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_gn)
print("Forma de R:", R_gn.shape)

plt.scatter(R_gn[:, 0], R_gn[:, 1], c='blue')
plt.title('Grafico de dispersion G/N')
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
plt.show()
plt.savefig("grafica2.eps", format='eps')
plt.close()

import numpy as np

A = X_train_pca
B = datos_generadosR['P/G'].values.reshape(-1, 1)

R_pg = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_pg[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_pg)
print("Forma de R:", R_pg.shape)

plt.scatter(R_pg[:, 0], R_pg[:, 1], c='green')
plt.title('Grafico de dispersion P/G')
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
plt.show()
plt.savefig("grafica3.eps", format='eps')
plt.close()

import numpy as np
import matplotlib.pyplot as plt

CG= R_cg
GN= R_gn
PG= R_pg


# Extrae las coordenadas x e y de cada arreglo
```

```
Ax = [x[0] for x in CG]
Ay = [x[1] for x in CG]

Bx = [x[0] for x in GN]
By = [x[1] for x in GN]

Cx = [x[0] for x in PG]
Cy = [x[1] for x in PG]



plt.figure(figsize=(8, 6))
plt.scatter(Ax, Ay, label='CG', color='blue')
plt.scatter(Bx, By, label='GN', color='green')
plt.scatter(Cx, Cy, label='PG', color='red')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de dispersión de CG, GN y PG')
plt.legend()
plt.grid(True)
# Guarda la gráfica como una imagen antes de mostrarla
plt.savefig('grafico_dispersion.png', format='png', dpi=300)

# Muestra la gráfica en pantalla
plt.show()

# Cierra la figura
plt.close()

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.scatter([x[0] for x in X_train_pca], [x[1] for x in X_train_pca],
color='blue')
plt.title('Visualización de datos estandarizados mediante PCA')
plt.xlabel('Componente principal 1')
plt.ylabel('Componente principal 2')
plt.grid(True)
plt.show()
plt.savefig("grafica5.eps", format='eps')
plt.close()

import numpy as np
cov_mat = np.cov(X_train_std)

eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues\n%s'% eigen_vals)

tot = sum(eigen_vals)
var_exp= [(i/tot)for i in
          sorted (eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

import matplotlib.pyplot as plt
```

```
from matplotlib.backends.backend_pdf import PdfPages
plt.bar(range(1, len(var_exp) + 1), var_exp, alpha=0.5, align='center',
        label='individual explained variance')
plt.step(range(1, len(cum_var_exp) + 1), cum_var_exp, where='mid',
         label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.savefig("explained-variance.png")
plt.show()


#Lista de tuplas
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
               for i in range(len(eigen_vals))]


#Ordena las tuplas

eigen_pairs.sort(key=lambda k: k[0], reverse=True)

w = np.hstack((eigen_pairs[0][1][:, np.newaxis],
               eigen_pairs[1][1][:, np.newaxis]))


print('Matriz W: \n', w)


print(np.size(w))
w.shape

#Estandarizamos
scaler  =  StandardScaler()
w_std = scaler.fit_transform(w)

import numpy as np

# Multiplicacion de C/G * W
A = w_std
B = datos_generadosR['C/G'].values.reshape(-1, 1)

R_w_cg = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_w_cg[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_w_cg)
print("Forma de R:", R_w_cg.shape)

plt.scatter(R_w_cg[:, 0], R_w_cg[:, 1], c='red')
plt.title('Grafico de dispersion C/G')
```

```
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
plt.show()
plt.savefig("grafica7.eps", format='eps')
plt.close()

import numpy as np

# Multiplicacion de G/N * W
A = w_std
B = datos_generadosR['G/N'].values.reshape(-1, 1)

R_w_gn = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_w_gn[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_w_gn)
print("Forma de R:", R_w_gn.shape)

plt.scatter(R_w_gn[:, 0], R_w_gn[:, 1], c='blue')
plt.title('Grafico de dispersion G/N')
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
plt.show()
plt.savefig("grafica8.eps", format='eps')
plt.close()

import numpy as np

# Multiplicacion de P/G * W
A = w_std
B = datos_generadosR['P/G'].values.reshape(-1, 1)

R_w_pg = np.zeros((47, 2))

# Realizar la multiplicación elemento por elemento
for i in range(A.shape[1]):  # Iterar sobre las columnas de X_train_pca
    R_w_pg[:, i] = A[:, i] * B[:, 0]

# Imprimir el resultado
print(R_w_pg)
print("Forma de R:", R_w_pg.shape)

plt.scatter(R_w_pg[:, 0], R_w_pg[:, 1], c='green')
plt.title('Grafico de dispersion P/G')
plt.xlabel('Columna 1')
plt.ylabel('Columna 2')
plt.grid(True)
```

```
plt.show()
plt.savefig("grafica9.eps", format='eps')
plt.close()

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages


# Función para filtrar puntos dentro de un rango en x e y
def filter_points(points, x_range, y_range):
    filtered_points = []
    for point in points:
        x, y = point
        if x_range[0] <= x <= x_range[1] and y_range[0] <= y <= y_range[1]:
            filtered_points.append(point)
    return filtered_points

# Filtrar los puntos según los rangos especificados
WCG_filtered = filter_points(R_w_cg, (-5, 5), (-4, 4))
WGN_filtered = filter_points(R_w_gn, (-5, 5), (-4, 4))
WPG_filtered = filter_points(R_w_pg, (-5, 5), (-4, 4))

# Extraer las coordenadas x e y de cada arreglo filtrado
WAx = [x[0] for x in WCG_filtered]
WAy = [x[1] for x in WCG_filtered]

WBx = [x[0] for x in WGN_filtered]
WBy = [x[1] for x in WGN_filtered]

WCx = [x[0] for x in WPG_filtered]
WCy = [x[1] for x in WPG_filtered]

# Graficar los puntos filtrados
fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(WAx, WAy, label='C/G', color='blue')
ax.scatter(WBx, WBy, label='G/N', color='green')
ax.scatter(WCx, WCy, label='P/G', color='red')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
#ax.set_title('Gráfico de dispersión de CG, GN y PG (Filtrado)')
ax.set_xlim(-5, 5)
ax.set_ylim(-4, 4)
ax.legend()
ax.grid(True)

# Mostrar la gráfica para verificar que se generó correctamente
plt.savefig("W.png")
plt.show()
plt.close()

from sklearn.cluster import KMeans
```

```
# Establecer el valor de n_init explícitamente para evitar la advertencia
kmeans = KMeans(n_clusters=3, random_state=0, n_init=10)
kmeans.fit(X_train_pca)

cluster_labels = kmeans.labels_

plt.figure(figsize=(8, 6))
for cluster_label in np.unique(cluster_labels):
    plt.scatter(X_train_pca[cluster_labels == cluster_label, 0],
                X_train_pca[cluster_labels == cluster_label, 1],
                label=f'Cluster {cluster_label}')

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.title('Visualización de clusters en PCA')
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.cluster import KMeans

combined_data = np.concatenate((WCG_filtered, WGN_filtered, WPG_filtered))

num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, n_init=10) # Establece
explícitamente n_init para evitar la advertencia
kmeans.fit(combined_data)
labels = kmeans.labels_

plt.figure(figsize=(8, 6))
plt.scatter(combined_data[:, 0], combined_data[:, 1], c=labels,
cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
marker='x', s=200, c='red', label='Centroides')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('K-Means Clustering')
plt.legend()
plt.grid(True)
plt.savefig("Clustering.png")
plt.show()

plt.close()

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Genera datos de muestra utilizando make_blobs
```

```
X, y = make_blobs(n_samples=150,
                  n_features=2,
                  centers=3,
                  cluster_std=0.5,
                  shuffle=True,
                  random_state=0)

# Aplica K-Means a los datos generados
kmeans = KMeans(n_clusters=3, n_init=10)
kmeans.fit(X)

# Combina los datos filtrados en un solo conjunto de datos
combined_data = np.concatenate((WCG_filtered, WGN_filtered, WPG_filtered))

# Aplica K-Means a los datos combinados
num_clusters = 3
kmeans_combined = KMeans(n_clusters=num_clusters, n_init=10)
kmeans_combined.fit(combined_data)
labels_combined = kmeans_combined.labels_

# Grafica los puntos y los centroides de los clusters para los datos
generados
plt.figure(figsize=(12, 6))

# Grafica los puntos y los centroides de los clusters para los datos
combinados
plt.subplot(1, 2, 2)
plt.scatter(combined_data[:, 0], combined_data[:, 1], c=labels_combined,
cmap='viridis')
#plt.scatter(kmeans_combined.cluster_centers_[:, 0],
kmeans_combined.cluster_centers_[:, 1], marker='x', s=200, c='magenta',
label='Centroides')
#plt.title('Clustering de datos combinados')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.grid(True)

# Scatter plots de los clusters y centroides para los datos combinados
plt.scatter(combined_data[labels_combined==0, 0],
            combined_data[labels_combined==0, 1],
            s=50, c='blue',
            marker='s', edgecolor='black',
            label='C/G')
plt.scatter(combined_data[labels_combined==1, 0],
            combined_data[labels_combined==1, 1],
            s=50, c='green',
            marker='o', edgecolor='black',
            label='G/N')
plt.scatter(combined_data[labels_combined==2, 0],
            combined_data[labels_combined==2, 1],
            s=50, c='red',
            marker='v', edgecolors='black',
```

```
            label='P/G')
plt.scatter(kmeans_combined.cluster_centers_[:, 0],
            kmeans_combined.cluster_centers_[:, 1],
            s=250, marker='*',
            c='cyan', edgecolor='black',
            label='centroids')
plt.legend(scatterpoints=1)
plt.grid(True)

plt.tight_layout()
plt.show()
plt.close()
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
datosEsR = pd.DataFrame({
    'WL': ['ASUS', 'BASE', 'BB', 'CHANK', 'CHE', 'CHP', 'CI', 'GP', 'HLSJP',
'HLSM', 'LGG', 'LLMAS', 'MP', 'NLS', 'SAP', 'SJMS', 'SUC', 'TC', 'Tosm'],
    'C/G': [-0.400093, -0.697184, 0.573534, 1.102661, 0.475120, -0.420393, -
0.615593, -0.662003, 0.871291, 0.871609, -0.601248, -0.501280, -0.501079,
0.745972, -1.783641, -0.507874, -0.480801, -0.662081, 1.173163],
    'G/N': [0.424203, 0.288838, 0.317337, -0.665333, -1.250608, 0.337671,
0.686446, 0.278849, -1.012132, -0.233622, 0.710727, 0.370157, 0.412883, -
0.986949, 0.688013, 0.249779, 0.521064, 0.491279, -0.522069],
    'P/G': [0.236123, 0.859110, 0.520017, 0.332811, -1.523581, 0.363944,
1.404858, -0.246928, -0.510184, 0.565983, 1.351302, 0.143797, 0.457168, -
0.790013, 0.825106, -0.463132, 0.226423, 0.701030, 0.566041]
})
numeric_columns = datosEsR.select_dtypes(include=['float64',
'int64']).columns

# Asegurarse de que hay columnas numéricas para escalar
if numeric_columns.size == 0:
    raise ValueError("No hay columnas numéricas para escalar.")

# Convertir los valores a sus valores absolutos
datosEsR[numeric_columns] = datosEsR[numeric_columns].abs()

# Escalar las columnas numéricas usando MinMaxScaler
scaler = MinMaxScaler()
datosEsR[numeric_columns] = scaler.fit_transform(datosEsR[numeric_columns])

print(datosEsR)

import pandas as pd

# DataFrame original
datosEsR = pd.DataFrame({
    'WL': ['ASUS', 'BASE', 'BB', 'CHANK', 'CHE', 'CHP', 'CI', 'GP', 'HLSJP',
'HLSM', 'LGG', 'LLMAS', 'MP', 'NLS', 'SAP', 'SJMS', 'SUC', 'TC', 'Tosm'],
    'C/G': [0.000000, 0.214731, 0.125360, 0.507802, 0.054228, 0.014672,
0.155759, 0.189303, 0.340572, 0.340802, 0.145391, 0.073136, 0.072991,
0.249994, 1.000000, 0.077902, 0.058334, 0.189360, 0.558759],
```

```
    'G/N': [0.187398, 0.054294, 0.082317, 0.424500, 1.000000, 0.102311,
0.445261, 0.044472, 0.765507, 0.000000, 0.469136, 0.134255, 0.176267,
0.740745, 0.446802, 0.015887, 0.282641, 0.253354, 0.283629],
    'P/G': [0.066913, 0.518424, 0.272666, 0.136988, 1.000000, 0.159552,
0.913955, 0.074744, 0.265539, 0.305980, 0.875141, 0.000000, 0.227116,
0.468346, 0.493779, 0.231438, 0.059883, 0.403855, 0.306022]
})

# Función para encontrar la columna con el valor máximo en cada fila
def get_max_column(row):
    max_value = max(row['C/G'], row['G/N'], row['P/G'])
    if row['C/G'] == max_value:
        return 'C/G'
    elif row['G/N'] == max_value:
        return 'G/N'
    else:
        return 'P/G'

# Aplicamos la función a cada fila del DataFrame
datosEsR['Max_Column'] = datosEsR.apply(get_max_column, axis=1)

print(datosEsR)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


# Normalizar los datos de `datosEsR` usando StandardScaler si es necesario
scaler = StandardScaler()
datosEs_norm = scaler.fit_transform(datosEsR[['C/G', 'G/N', 'P/G']])

# Aplicar K-means clustering para encontrar grupos similares usando los datos
normalizados
num_clusters = 6
kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)  #
Establece n_init explícitamente
clusters = kmeans.fit_predict(datosEs_norm)

# Agregar los clusters al DataFrame original
datosEsR['Cluster'] = clusters

# Visualizar los resultados de los clusters
plt.figure(figsize=(10, 6))
colors = ['red', 'green', 'blue', 'purple', 'orange', 'cyan']

for cluster_num in range(num_clusters):
    subset = datosEsR[datosEsR['Cluster'] == cluster_num]
    plt.scatter(subset['C/G'], subset['G/N'], color=colors[cluster_num],
label=f'Cluster {cluster_num}', alpha=0.6)
    # Añadir etiquetas de nombre de aceite esencial
    for i, row in subset.iterrows():
```

```
        plt.text(row['C/G'], row['G/N'], row['WL'], fontsize=8)

plt.title('Clusters de Quimiotipos de Aceites Esenciales')
plt.xlabel('C/G')
plt.ylabel('G/N')
plt.legend()
plt.grid(True)
plt.savefig("clusters_quimiotipos.png")
plt.show()

# Mostrar los nombres de los aceites esenciales en cada cluster
for cluster_num in range(num_clusters):
    print(f"\nCluster {cluster_num}:")
    print(datosEsR[datosEsR['Cluster'] == cluster_num]['WL'].values)

# Opcional: Graficar los perfiles de características para cada cluster
for cluster_num in range(num_clusters):
    plt.figure(figsize=(10, 6))
    subset = datosEsR[datosEsR['Cluster'] == cluster_num]
    for index, row in subset.iterrows():
        # Graficar las características `C/G`, `G/N`, `P/G`
        plt.plot(['C/G', 'G/N', 'P/G'], row[['C/G', 'G/N', 'P/G']],
alpha=0.7)

    plt.title(f'Cluster {cluster_num} ({len(subset)} aceites esenciales)')
    plt.xlabel('Características')
    plt.ylabel('Valor Escalado')
    plt.savefig(f"cluster_{cluster_num}_perfil.png")
    plt.show()
```