

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Inteligencia artificial avanzada para la ciencia de datos I
TC3006C, Grupo 101



Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Jorge Adolfo Ramírez Uresti

**Momento de Retroalimentación: Módulo 2 Uso de framework
o biblioteca de aprendizaje máquina para la implementación
de una solución. (Portafolio Implementación)**

Adolfo Sebastián González Mora | A01754412

9 de Septiembre 2024

Introducción

El objetivo de este proyecto es desarrollar un modelo de regresión logística optimizado mediante la selección de características y preprocesamiento de datos para predecir la calidad del aire. A través de esta implementación, se busca profundizar en los conceptos fundamentales del aprendizaje automático, empleando técnicas como la Eliminación Recursiva de Características (RFE) y la normalización de datos con MinMaxScaler, para mejorar el rendimiento del modelo.

La regresión logística es un algoritmo ampliamente utilizado en clasificación binaria debido a su simplicidad, interpretabilidad y efectividad en la predicción de probabilidades. Este tipo de modelo es particularmente útil en situaciones donde el objetivo es clasificar resultados en dos categorías, como en este caso, la predicción de si la calidad del aire es buena o mala. Aplicaciones de este tipo de modelos se encuentran en áreas como la medicina, la predicción de fraudes financieros y la evaluación de riesgos en el ámbito medioambiental.

Este proyecto abarca todas las etapas clave del desarrollo del modelo, incluyendo:

- **Preprocesamiento de datos:** Asegurando que el conjunto de datos esté normalizado y en un formato adecuado para el análisis, utilizando MinMaxScaler para escalar los valores entre 0 y 1.
- **Selección de características:** Implementando la técnica de RFE (Eliminación Recursiva de Características) para seleccionar las características más relevantes, mejorando la precisión y eficiencia del modelo.
- **Entrenamiento del modelo:** Desarrollando un modelo de regresión logística que se ajusta a los datos seleccionados y escalados, utilizando una cuadrícula de búsqueda para optimizar los hiperparámetros.
- **Evaluación del modelo:** Aplicando métricas como la precisión, el recall, la puntuación F1 y la matriz de confusión para evaluar el rendimiento del modelo en la predicción de la calidad del aire.

A lo largo de este documento, se explicarán cada uno de estos pasos con detalle, proporcionando tanto un enfoque teórico como ejemplos prácticos que ilustran la creación y validación del modelo. Además, se presentarán los resultados obtenidos, y se discutirán posibles mejoras en la implementación para su uso en aplicaciones futuras.

Desarrollo del Proyecto

El desarrollo de este proyecto se organizó de manera modular, con varios archivos Python (.py) que trabajan en conjunto para llevar a cabo las tareas de preprocesamiento, entrenamiento y predicción en el ámbito del aprendizaje automático. Cada archivo tiene una función clara y específica, lo que facilita la mantenibilidad y reutilización del código. Este enfoque modular refleja las mejores prácticas de desarrollo y asegura una organización eficiente.

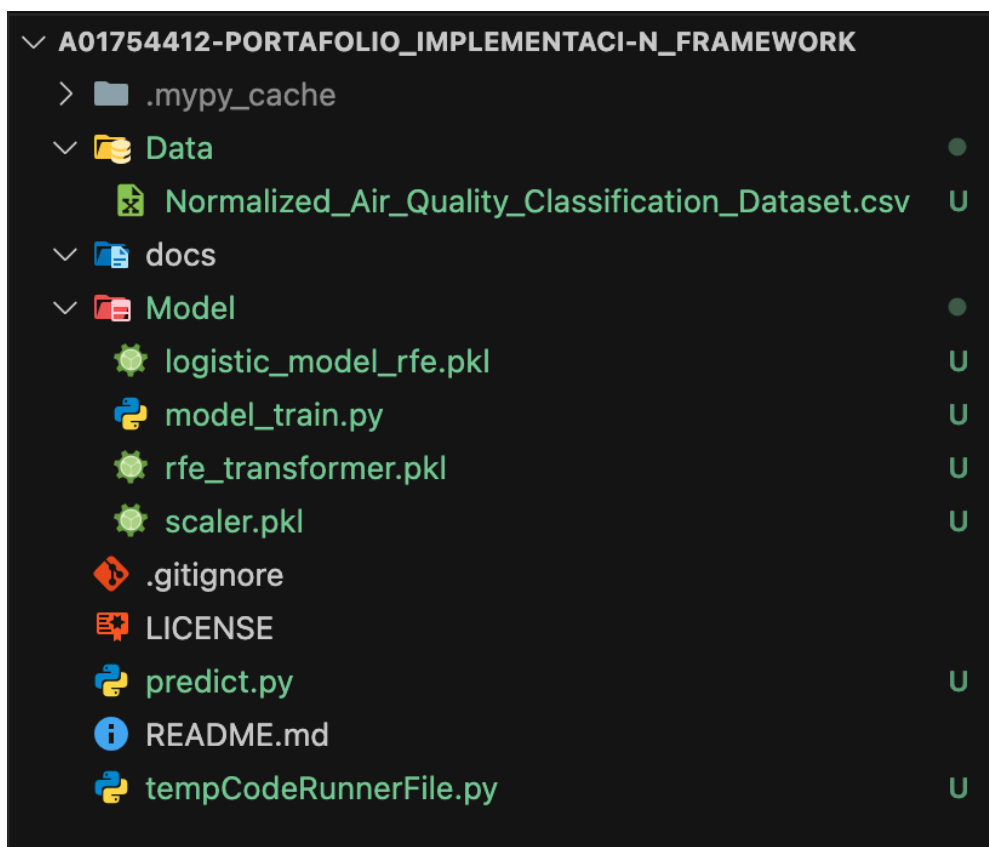
La estructura del proyecto es la siguiente:

- **Data:** Contiene el conjunto de datos principal utilizado en este proyecto, en este caso, el archivo `Normalized_Air_Quality_Classification_Dataset.csv`. Este dataset ha sido normalizado previamente y es fundamental para el entrenamiento y evaluación del modelo de regresión logística.
- **docs:** Carpeta destinada a la documentación del proyecto. Aquí se almacena cualquier referencia adicional o información relevante para comprender y utilizar el código.
- **Model:** Este directorio se enfoca en la implementación y entrenamiento del modelo de regresión logística, junto con las herramientas de preprocesamiento. Contiene los siguientes archivos:
 - **logistic_model_rfe.pkl:** Archivo que guarda el modelo de regresión logística entrenado, listo para ser utilizado en predicciones futuras sin necesidad de reentrenamiento.
 - **rfe_transformer.pkl:** Contiene el objeto RFE (Eliminación Recursiva de Características), utilizado para seleccionar las características más importantes del conjunto de datos durante el entrenamiento.
 - **scaler.pkl:** Este archivo guarda el objeto MinMaxScaler, que fue empleado para normalizar los datos de entrada. Es fundamental para garantizar que cualquier nuevo dato sea procesado de manera consistente con los datos de entrenamiento.
 - **model_train.py:** Script encargado de la construcción, entrenamiento y validación del modelo de regresión logística, incluyendo la selección de características y escalado de datos.
- **predict.py:** Este archivo contiene el código necesario para realizar predicciones utilizando el modelo entrenado. Carga los archivos .pkl guardados (modelo, RFE y

escalador) y los aplica a nuevos datos para predecir la calidad del aire, ya sea buena o mala.

- **README.md:** Proporciona una descripción general del proyecto, incluyendo instrucciones sobre cómo ejecutar el código, información sobre los archivos .pkl generados, y detalles adicionales que permiten a otros usuarios comprender y reutilizar el proyecto.

Además, el proyecto incluye archivos importantes como .gitignore, que se utiliza para excluir ciertos archivos o directorios del control de versiones, y LICENSE, que detalla las condiciones bajo las cuales se puede utilizar y distribuir el código.



Explicación del código

En el archivo `model_train.py`, me encargué de crear, entrenar y optimizar un modelo de regresión logística para predecir la calidad del aire usando distintas características ambientales, el primer paso fue cargar y preprocesar los datos, asegurándome de que estuvieran en el formato adecuado. Esto incluyó la normalización de las características y el manejo de valores faltantes, para que el modelo pudiera procesar correctamente la información.

Después de preparar los datos, utilicé Eliminación Recursiva de Características (RFE) para seleccionar las 10 características más relevantes, esto fue fundamental, ya que permite que el modelo se enfoque en las variables que más influyen en la predicción, mejorando así su rendimiento y reduciendo la complejidad de los datos.

Luego, entrené el modelo de regresión logística, ajustando sus hiperparámetros mediante GridSearchCV, esta técnica me permitió probar diferentes combinaciones de hiperparámetros, como el valor de regularización C, el tipo de penalización y el solver, para encontrar la mejor configuración que maximizara la precisión del modelo. Esta validación cruzada me aseguró que el modelo no solo funcionara bien en el conjunto de entrenamiento, sino también en datos que no había visto antes.

Una vez optimizado el modelo, lo evalué usando el conjunto de validación, calculando métricas clave como la precisión, el recall, la F1 y la matriz de confusión, esto me permitió ver qué tan bien el modelo podía predecir la calidad del aire y si estaba funcionando como esperaba.

Después, realicé una evaluación final con el conjunto de prueba, que contiene datos completamente nuevos para el modelo, evalué nuevamente su rendimiento para asegurarme de que pudiera generalizar bien y no estaba sobreajustado a los datos de entrenamiento.

Por último, guardé el modelo entrenado, el objeto RFE y el escalador MinMaxScaler en archivos .pkl utilizando pickle, esto me permite reutilizar el modelo y las transformaciones de los datos sin necesidad de volver a entrenarlo o recalcular las transformaciones en el futuro, facilitando las predicciones con nuevos datos.

En resumen, el código implementa un modelo de regresión logística para predecir la calidad del aire, optimizándolo y evaluándolo de manera rigurosa, con el fin de asegurar que sea preciso y pueda ser utilizado en el futuro sin necesidad de reentrenamiento.

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

```
from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_selection import RFE

from sklearn.impute import SimpleImputer

import pickle


file_path =
'/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Data/Normalize
d_Air_Quality_Classification_Dataset.csv'

data = pd.read_csv(file_path)


X = data.drop('Air_Quality', axis=1)

y = data['Air_Quality']

X_numeric = X.select_dtypes(include=[np.number])

imputer = SimpleImputer(strategy='median')

X_numeric = imputer.fit_transform(X_numeric)


X_train, X_temp, y_train, y_temp = train_test_split(
    X_numeric, y, test_size=0.3, random_state=42)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42)


scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_val = scaler.transform(X_val)

X_test = scaler.transform(X_test)
```

```
log_reg = LogisticRegression(random_state=42, max_iter=100)

rfe = RFE(log_reg, n_features_to_select=10)

X_train_rfe = rfe.fit_transform(X_train, y_train)

X_val_rfe = rfe.transform(X_val)

X_test_rfe = rfe.transform(X_test)

param_grid = [

    {'C': [0.01, 1, 10, 100], 'penalty': ['l2'], 'solver': ['lbfgs'], 'class_weight': [None, 'balanced']},

    {'C': [0.01, 1, 10, 100], 'penalty': ['l1'], 'solver': ['liblinear'], 'class_weight': [None, 'balanced']}

]

grid_search = GridSearchCV(LogisticRegression(random_state=42, max_iter=100),

                           param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train_rfe, y_train)

best_model = grid_search.best_estimator_

print(f"Mejor parámetro C: {grid_search.best_params_['C']}")

print(f"Mejor penalización: {grid_search.best_params_['penalty']}")

print(f"Mejor solver: {grid_search.best_params_['solver']}")

print(f"Mejor balanceo de clases: {grid_search.best_params_['class_weight']}")
```

```
y_pred_val = best_model.predict(X_val_rfe)

accuracy_val = accuracy_score(y_val, y_pred_val)

precision_val = precision_score(y_val, y_pred_val, average='macro')

recall_val = recall_score(y_val, y_pred_val, average='macro')

f1_val = f1_score(y_val, y_pred_val, average='macro')

print(f"Evaluación en el conjunto de validación:")

print(f"Precisión (Accuracy): {accuracy_val:.16f}")

print(f"Precisión: {precision_val:.16f}")

print(f"Recall: {recall_val:.16f}")

print(f"F1 Score: {f1_val:.16f}")

print("Matriz de confusión:\n", confusion_matrix(y_val, y_pred_val))

y_pred_test = best_model.predict(X_test_rfe)

accuracy_test = accuracy_score(y_test, y_pred_test)

precision_test = precision_score(y_test, y_pred_test, average='macro')

recall_test = recall_score(y_test, y_pred_test, average='macro')

f1_test = f1_score(y_test, y_pred_test, average='macro')

print(f"\nEvaluación en el conjunto de prueba:")

print(f"Precisión (Accuracy): {accuracy_test:.16f}")
```



```

print(f"Precisión: {precision_test:.16f}")

print(f"Recall: {recall_test:.16f}")

print(f"F1 Score: {f1_test:.16f}")


print("Matriz de confusión (Prueba):\n", confusion_matrix(y_test, y_pred_test))

with
open('/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/rfe
_transformer.pkl', 'wb') as rfe_file:

    pickle.dump(rfe, rfe_file)


with
open('/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/log
istic_model_rfe.pkl', 'wb') as model_file:

    pickle.dump(best_model, model_file)


with
open('/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/sca
ler.pkl', 'wb') as scaler_file:

    pickle.dump(scaler, scaler_file)

```

NOTA: Al momento de cargar el código decidí eliminar los comentarios para que no se hiciera tan extenso el documento.

En el archivo `predict.py`, me encargué de cargar el modelo de regresión logística ya entrenado y sus componentes asociados (el objeto RFE y el escalador `MinMaxScaler`) para realizar predicciones sobre la calidad del aire utilizando nuevas mediciones. El primer paso fue importar las bibliotecas necesarias, como NumPy para la manipulación de matrices y pickle para cargar los archivos `.pkl` que contienen el modelo y los objetos que guardé durante el entrenamiento.

Luego, cargué el modelo y los componentes usando pickle; específicamente, cargué:

- El archivo del modelo de regresión logística (`logistic_model_rfe.pkl`), que me permite reutilizar el modelo sin tener que entrenarlo nuevamente.

- El archivo del objeto RFE (`rfe_transformer.pkl`), que se utilizó para seleccionar las características más importantes durante el entrenamiento, esto es clave, ya que asegura que las mismas características seleccionadas en el entrenamiento se utilicen para hacer predicciones con nuevos datos.
- El archivo del escalador MinMaxScaler (`scaler.pkl`), que se utilizó para normalizar las características de entrada, esto es importante para garantizar que los nuevos datos tengan el mismo formato y escala que los datos de entrenamiento.

Una vez que cargué estos componentes, creé un conjunto de características utilizando un arreglo de NumPy; este arreglo contiene mediciones que corresponden a una situación en la que la calidad del aire es buena o mala, dependiendo de las características que se ingresen.

El siguiente paso fue escalar las características usando el escalador previamente cargado, para asegurar que estén en el mismo rango que los datos que se usaron para entrenar el modelo, luego, utilicé el objeto RFE para seleccionar las características más relevantes. Esto garantiza que el modelo trabaje solo con las características que más influyen en la predicción.

Finalmente, realicé la predicción utilizando el modelo de regresión logística cargado; el modelo devuelve un valor binario: 1 para indicar que la calidad del aire es buena, o 0 para indicar que es mala, dependiendo del valor predicho, imprimí un mensaje indicando si la calidad del aire es buena o mala.

Este archivo es fundamental para realizar predicciones rápidas sin necesidad de reentrenar el modelo, aprovechando los objetos y el modelo que ya había guardado previamente, la estructura modular del código permite una reutilización eficiente de los componentes y facilita la integración de nuevos datos para predicciones futuras.

```
import numpy as np
import pickle

model_path =
'/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/logistic_
_model_rfe.pkl'
rfe_path =
'/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/rfe_tran
sformer.pkl'
scaler_path =
'/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/Model/scaler.p
kl'

with open(model_path, 'rb') as model_file:
    model = pickle.load(model_file)

with open(rfe_path, 'rb') as rfe_file:
```

```

rfe = pickle.load(rfe_file)

with open(scaler_path, 'rb') as scaler_file:
    scaler = pickle.load(scaler_file)

features = np.array([[0.2343438717484787, 0.9629080691788374, 0.7546155343748744,
0.8656603791107658, 0.8959211491318471, 0.7565152164393738, 0.7924601300347778,
0.2547923734351622, 0.0489159263896771, 0.0673856481784072, 0.0539250088719914,
0.2218700464732332, 0.0, 0.0, 0.0, 0.2333098756530392, 0.036363089175756,
0.3485216674026032, 0.210213941270986, 0.0877438661391807,
0.5694815607760904, 0.7717871554102588, 0.80439440249228, 0.8875362190202837,
0.0459683797598271, 0.2312954534461369, 0.2484252166252645, 0.2405081657963307,
0.8260394209253437, 0.773988137778487, 0.3556228133496051, 0.0,
0.3070491926854841]])

"""
Este bloque crea un arreglo NumPy llamado 'features', que contiene un conjunto de
características escaladas para hacer una predicción con el modelo de regresión
logística.

Los valores en este arreglo representan mediciones que corresponden a una situación
en la que la calidad del aire es mala.

Este conjunto de características será utilizado por el modelo para predecir si la
calidad del aire es mala (0) o buena (1).
"""

"""

features = np.array([[0.110673923409422, 0.9148332136747168, 0.8909984273579148,
0.8995179254995496, 0.8935876314134166, 0.8847968420808114, 0.8542722204215969,
0.4453182563203886, 0.12078067892359, 0.0430389068572398, 0.0813371111351689,
0.0703389359430209, 0.0, 0.0, 0.0, 0.5764024584471832, 0.5598051579455686,
0.2637290981774702, 0.2764028421549014, 0.1688909234253647,
0.2479540151231041, 0.8191677103269474, 0.844713039794972, 0.8875362190202837,
0.0459683797598271, 0.10069202866066, 0.1197493157112603, 0.1264384885344016,
0.5414854090473756, 0.858758586139794, 0.6429919760720377, 0.0,
0.5244725942753523]])

"""

features_scaled = scaler.transform(features)
features_selected = rfe.transform(features_scaled)
prediction = model.predict(features_selected)

```

```
if prediction == 1:
    print("La calidad del aire es Buena.")
else:
    print("La calidad del aire es Mala.")
```

Manual de usuario para que se puedan hacer predicciones

- Paso 1: Abrir el Archivo `predict.py`

Abre el archivo `predict.py` en tu entorno de desarrollo, asegúrate de estar en la pestaña correcta para que puedas ejecutar el código de predicción sin problemas.

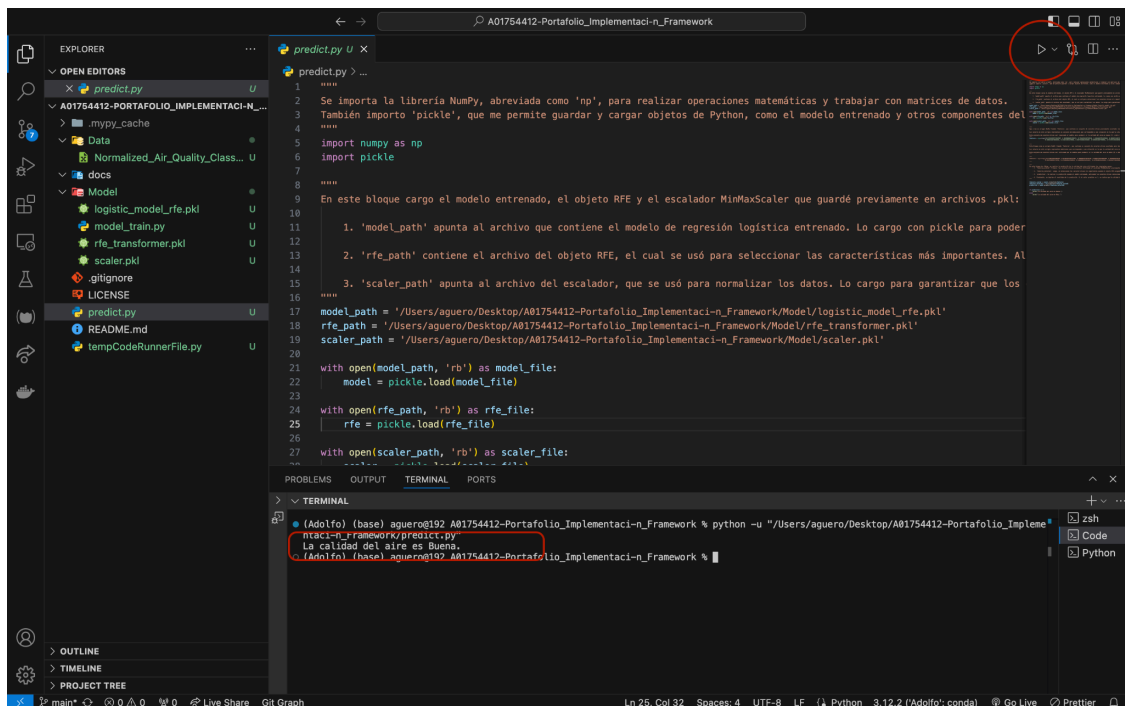
- Paso 2: Ejecutar el Código

Una vez que estés en la pestaña de `predict.py`, localiza el botón de Run o Ejecutar en tu entorno de desarrollo (puede variar dependiendo del editor que estés utilizando, como VSCode, PyCharm o Jupyter Notebook). Haz clic en el botón asegurándote de que la pestaña de `predict.py` esté activa, esto es importante para evitar que se ejecute un código incorrecto desde otra pestaña.

Al presionar Run, el código en `predict.py` cargará automáticamente el modelo entrenado, normalizará las características de calidad del aire que especificaste, y utilizará el modelo de regresión logística para hacer una predicción.

- Paso 3: Ver los Resultados

Después de la ejecución, el código imprimirá en la consola si la calidad del aire es "buena" o "mala", según la predicción realizada. Asegúrate de revisar la consola para ver el resultado.



Si deseas probar el modelo con diferentes características de calidad del aire, puedes hacerlo fácilmente modificando el arreglo de features en el archivo `predict.py`.

- Paso 1: Abrir el Archivo `predict.py`

Abre el archivo `predict.py` en tu entorno de desarrollo.

- Paso 2: Localizar el Arreglo features

Dentro del código, encontrarás una sección que define un arreglo llamado `features`, este arreglo contiene los valores numéricos que representan las diferentes características del aire, como el nivel de partículas, la temperatura, la humedad, entre otros.

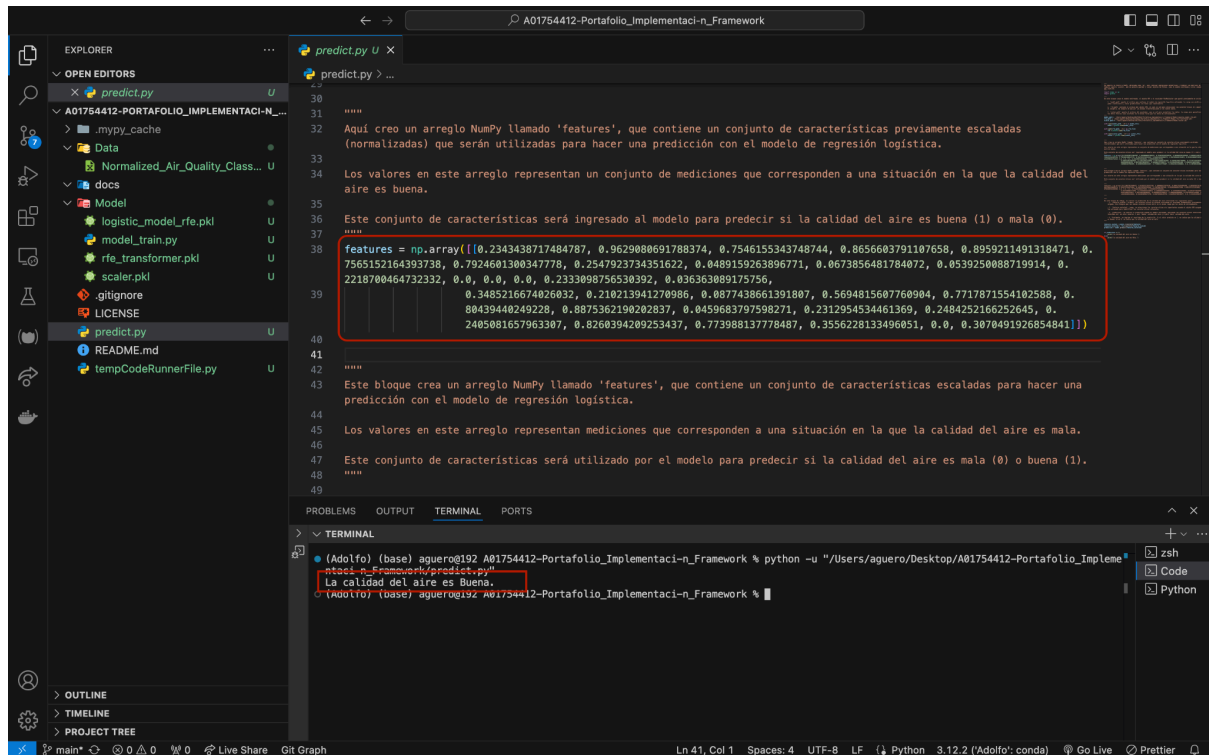
- Paso 3: Modificar las Características

Para probar el modelo con un nuevo conjunto de datos, reemplaza los valores dentro del arreglo `features` con las nuevas características que deseas evaluar, asegúrate de que los valores sigan el mismo orden y formato que los actuales, ya que cada posición en el arreglo corresponde a una característica específica del aire.

- Paso 4: Ejecutar el Código

Después de haber ingresado las nuevas características, guarda el archivo y vuelve a ejecutar el código como se describió anteriormente. El modelo utilizará las nuevas características para hacer una predicción, y el resultado se imprimirá en la consola.

(Predicción de calidad de aire buena)



The screenshot shows a VS Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'A01754412-Portafolio_Implementaci-n_Framework' with files like 'predict.py', 'logistic_model_rfe.pkl', 'model_train.py', 'rfe_transformer.pkl', 'scaler.pkl', '.gitignore', 'LICENSE', 'README.md', and 'tempCodeRunnerFile.py'. The code editor shows the 'predict.py' file with the following code:

```
30 """
31 Aquí creo un arreglo NumPy llamado 'features', que contiene un conjunto de características previamente escaladas
32 (normalizadas) que serán utilizadas para hacer una predicción con el modelo de regresión logística.
33
34 Los valores en este arreglo representan un conjunto de mediciones que corresponden a una situación en la que la calidad del
35 aire es buena.
36
37 Este conjunto de características será ingresado al modelo para predecir si la calidad del aire es buena (1) o mala (0).
38 """
39 features = np.array([[0.2343438717484787, 0.9629880691788374, 0.7546155343748744, 0.8656603791107658, 0.8959211491318471, 0.
7565152164393738, 0.7924601300347778, 0.2547923734351622, 0.0489159263896771, 0.0673856481784072, 0.0539250088719914, 0.
2218700464732332, 0.0, 0.0, 0.0, 0.2333098756530392, 0.036363089175756,
0.3485216674826032, 0.210213941270986, 0.8877438661391807, 0.5694815607760904, 0.7717871554102588, 0.
80439448249228, 0.8875362190202837, 0.0459683797598271, 0.2312954534461369, 0.2484252166252645, 0.
2485081657963307, 0.8260394289253437, 0.773988137778487, 0.3556228133496051, 0.0, 0.3070491926854841]])
40
41
42 """
43 Este bloque crea un arreglo NumPy llamado 'features', que contiene un conjunto de características escaladas para hacer una
44 predicción con el modelo de regresión logística.
45
46 Los valores en este arreglo representan mediciones que corresponden a una situación en la que la calidad del aire es mala.
47
48 Este conjunto de características será utilizado por el modelo para predecir si la calidad del aire es mala (0) o buena (1).
49 """
```

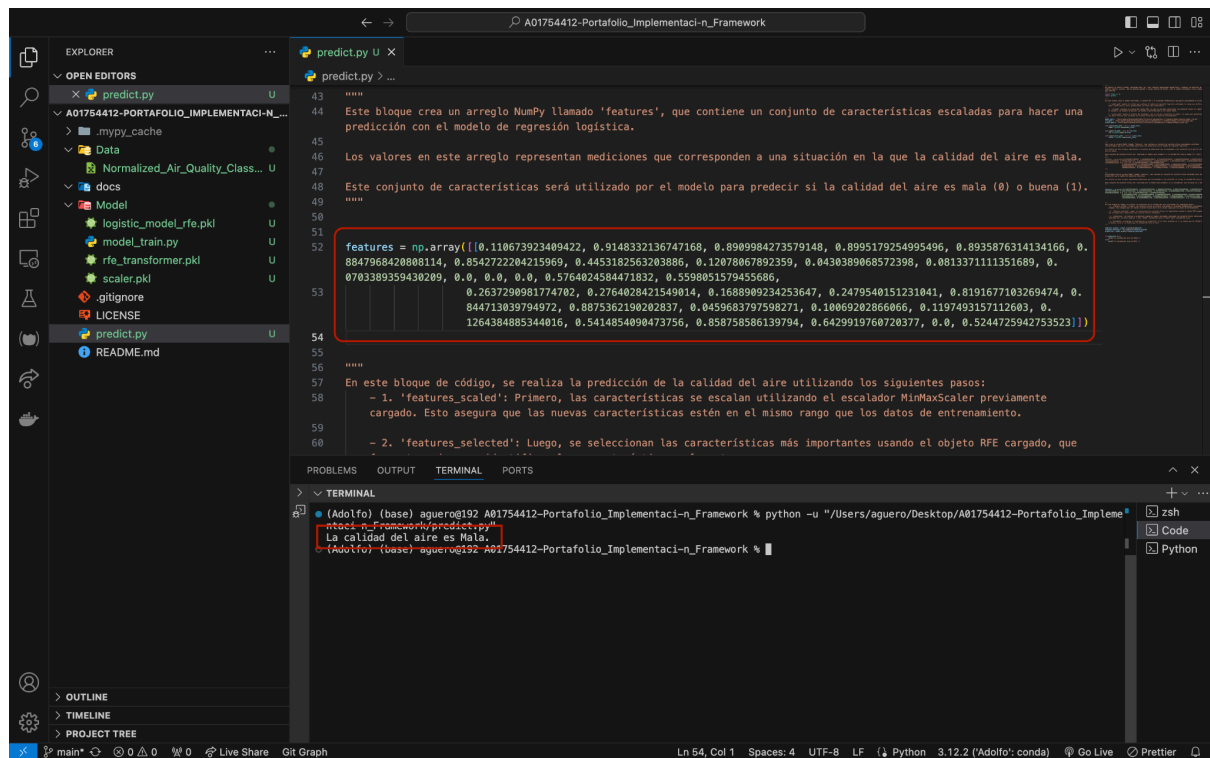
The terminal output shows the following command and result:

```
python -u "/Users/aguero/Desktop/A01754412-Portafolio_Implementaci-n_Framework/predict.py"
La calidad del aire es Buena.
```

Con base en los datos ingresados, el modelo ha predicho que la calidad del aire es buena esta predicción se fundamenta en el análisis de las características proporcionadas, lo que sugiere que las condiciones actuales del aire cumplen con los criterios establecidos para considerarse de buena calidad.

(Predicción de calidad de aire mala)

Si utilizamos otros datos, como los que se encuentran comentados en el código, obtendremos una predicción de mala calidad del aire, estos datos, que representan mediciones distintas, darán como resultado una predicción donde el modelo determinará que la calidad del aire no es óptima, como se muestra a continuación.



Manual de usuario para evaluar el modelo

- Paso 1: Abrir el Archivo `model_train.py`

Abre el archivo `model_train.py` en tu entorno de desarrollo. Asegúrate de estar en la pestaña correcta para que puedas ejecutar el código de entrenamiento sin problemas.

- Paso 2: Ejecutar el Código

Una vez que tengas abierto `model_train.py`, localiza el botón de Run o Ejecutar en tu entorno de desarrollo (esto puede variar dependiendo del editor que estés utilizando, como VSCode, PyCharm o Jupyter Notebook), asegúrate de que la pestaña de `model_train.py` esté activa antes de presionar Run para evitar ejecutar un archivo incorrecto desde otra pestaña.

Al presionar Run, el código en `model_train.py` cargará automáticamente los datos de calidad del aire, preprocesará las características, seleccionará las más importantes utilizando RFE, entrenará el modelo de regresión logística, y evaluará su rendimiento.

- Paso 3: Ver los Resultados

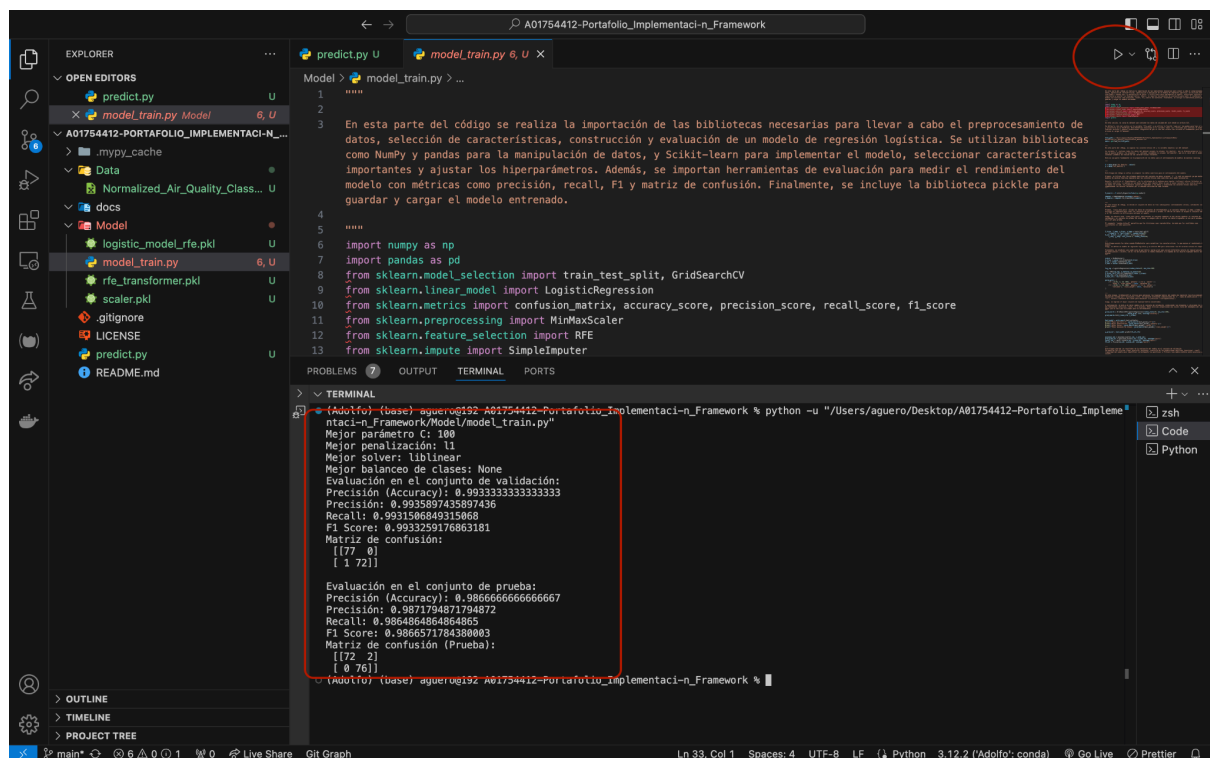
Después de ejecutar el código, los resultados del modelo aparecerán en la consola. Podrás ver las métricas de evaluación como la precisión (accuracy), el recall, el F1 Score, y la matriz de confusión, estas métricas te indicarán el rendimiento del modelo entrenado, asegúrate de revisar la consola para ver los detalles completos.

- Paso 4: Guardar el Modelo

Al finalizar la ejecución del código, el modelo entrenado y sus componentes, como el escalador y el objeto RFE, se guardarán automáticamente en la carpeta Model con los siguientes nombres:

- **logistic_model_rfe.pkl**: Contiene el modelo de regresión logística entrenado.
- **rfe_transformer.pkl**: Contiene el objeto RFE que selecciona las características más relevantes.
- **scaler.pkl**: Contiene el escalador MinMaxScaler utilizado para normalizar las características.

Con estos archivos guardados, podrás reutilizarlos para realizar predicciones futuras sin tener que reentrenar el modelo.



The screenshot shows a VS Code editor with a project named 'A01754412-Portfolio_Implementaci-n_Framework'. The Explorer panel on the left shows the project structure, including a 'Model' folder containing 'logistic_model_rfe.pkl', 'rfe_transformer.pkl', and 'scaler.pkl'. The main editor displays a Python script 'model_train.py' with comments in Spanish and Python code for data preprocessing and model training. The Terminal panel at the bottom shows the output of running the script, which includes the best parameter C (100), the best solver (liblinear), the class balance (None), and evaluation metrics for both the training and testing sets. The training set metrics are: Precision (Accuracy): 0.9933333333333333, Precision: 0.9935897435897436, Recall: 0.9931506849315068, F1 Score: 0.9933259176863181, and a confusion matrix. The testing set metrics are: Precision (Accuracy): 0.9866666666666667, Precision: 0.9871794871794872, Recall: 0.9864864864864865, F1 Score: 0.9866571784380083, and a confusion matrix.

```
Model > model_train.py > ...
1  """
2
3  En esta parte del código se realiza la importación de las bibliotecas necesarias para llevar a cabo el preprocesamiento de
4  datos, selección de características, construcción y evaluación de un modelo de regresión logística. Se utilizan bibliotecas
5  como NumPy y pandas para la manipulación de datos, y Scikit-learn para implementar el modelo, seleccionar características
6  importantes y ajustar los hiperparámetros. Además, se importan herramientas de evaluación para medir el rendimiento del
7  modelo con métricas como precisión, recall, F1 y matriz de confusión. Finalmente, se incluye la biblioteca pickle para
8  guardar y cargar el modelo entrenado.
9
10 """
11 import numpy as np
12 import pandas as pd
13 from sklearn.model_selection import train_test_split, GridSearchCV
14 from sklearn.linear_model import LogisticRegression
15 from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
16 from sklearn.preprocessing import MinMaxScaler
17 from sklearn.feature_selection import RFE
18 from sklearn.impute import SimpleImputer
19
20 # Cargar los datos
21 data = pd.read_csv('Data/Normalized_Air_Quality_Classification.csv')
22
23 # Preprocesamiento de datos
24 # Seleccionar características relevantes
25 rfe = RFE(LogisticRegression(), 10)
26 rfe.fit(data[['PM2.5', 'PM10', 'O3', 'NO2', 'CO', 'SO2', 'Temp', 'Humidity', 'WindSpeed', 'WindDir']])
27
28 # Seleccionar las características más relevantes
29 selected_features = data.columns[rfe.support_]
30
31 # Escalar las características
32 scaler = MinMaxScaler()
33 data[selected_features] = scaler.fit_transform(data[selected_features])
34
35 # Dividir los datos en conjuntos de entrenamiento y prueba
36 X_train, X_test, y_train, y_test = train_test_split(data[selected_features], data['Class'], test_size=0.2, random_state=42)
37
38 # Crear un modelo de regresión logística
39 model = LogisticRegression(solver='liblinear')
40
41 # Entrenar el modelo
42 model.fit(X_train, y_train)
43
44 # Guardar el modelo entrenado
45 pickle.dump(model, open('Model/logistic_model_rfe.pkl', 'wb'))
46
47 # Guardar el objeto RFE
48 pickle.dump(rfe, open('Model/rfe_transformer.pkl', 'wb'))
49
50 # Guardar el escalador
51 pickle.dump(scaler, open('Model/scaler.pkl', 'wb'))
52
53 # Evaluar el modelo
54 # Matriz de confusión
55 cm_train = confusion_matrix(y_train, model.predict(X_train))
56 cm_test = confusion_matrix(y_test, model.predict(X_test))
57
58 # Métricas de evaluación
59 accuracy_train = accuracy_score(y_train, model.predict(X_train))
60 accuracy_test = accuracy_score(y_test, model.predict(X_test))
61 precision_train = precision_score(y_train, model.predict(X_train))
62 precision_test = precision_score(y_test, model.predict(X_test))
63 recall_train = recall_score(y_train, model.predict(X_train))
64 recall_test = recall_score(y_test, model.predict(X_test))
65 f1_train = f1_score(y_train, model.predict(X_train))
66 f1_test = f1_score(y_test, model.predict(X_test))
67
68 # Imprimir los resultados
69 print("Mejor parámetro C: 100")
70 print("Mejor penalización: l1")
71 print("Mejor solver: liblinear")
72 print("Mejor balanceo de clases: None")
73 print("Evaluación en el conjunto de validación:")
74 print("Precisión (Accuracy): 0.9933333333333333")
75 print("Precisión: 0.9935897435897436")
76 print("Recall: 0.9931506849315068")
77 print("F1 Score: 0.9933259176863181")
78 print("Matriz de confusión:")
79 print(cm_train)
80
81 print("Evaluación en el conjunto de prueba:")
82 print("Precisión (Accuracy): 0.9866666666666667")
83 print("Precisión: 0.9871794871794872")
84 print("Recall: 0.9864864864864865")
85 print("F1 Score: 0.9866571784380083")
86 print("Matriz de confusión (Prueba):")
87 print(cm_test)
```


Análisis de resultados

Mejor Parámetro C: 100

- Este valor corresponde al parámetro de regularización C que el modelo de regresión logística ha seleccionado como el óptimo mediante GridSearchCV, un valor más alto de C indica que el modelo aplica menos regularización, lo que significa que el modelo se ajusta más a los datos de entrenamiento.

Mejor Penalización: l1

- El modelo utiliza la penalización l1, que corresponde a la regularización Lasso, este tipo de regularización tiende a reducir los coeficientes de algunas características a cero, lo que hace que sea útil para la selección de características en problemas de clasificación.

Mejor Solver: liblinear

- El solver liblinear es un algoritmo de optimización que funciona bien con problemas de clasificación pequeños y medianos, y es el más adecuado cuando se utiliza la regularización l1.

Mejor Balanceo de Clases: None

- Esto indica que no se aplicó un balanceo de clases específico en el modelo, significa que el modelo no ajustó sus pesos para contrarrestar posibles desequilibrios en las clases del conjunto de datos.

```
Mejor parámetro C: 100
Mejor penalización: l1
Mejor solver: liblinear
Mejor balanceo de clases: None
```

Evaluación en el Conjunto de Validación

Precisión (Accuracy): 0.9933

- Esto indica que el 99.33% de las predicciones realizadas en el conjunto de validación fueron correctas, la precisión (accuracy) es una métrica global que refleja qué tan bien el modelo predice correctamente ambas clases.

Precisión: 0.9935

- La precisión mide la proporción de predicciones positivas correctas (predicciones de buena calidad del aire), un valor de 0.9935 significa que el 99.35% de las veces que el modelo predijo que la calidad del aire era buena, acertó.

Recall: 0.9932

- El recall mide la capacidad del modelo para identificar correctamente los casos positivos (es decir, cuando la calidad del aire es buena), un valor de 0.9932 indica que el 99.32% de los casos positivos fueron correctamente identificados.

F1 Score: 0.9933

- El F1 Score es la media armónica entre la precisión y el recall, y en este caso tiene un valor de 0.9933, esta métrica equilibra ambos factores y es útil cuando es importante tanto identificar correctamente los positivos como minimizar los falsos positivos.

Matriz de Confusión (Validación):

- [[77, 0], [1, 72]]

La matriz de confusión muestra el rendimiento del modelo:

- **77 verdaderos negativos:** El modelo predijo correctamente que la calidad del aire era mala en 77 ocasiones.
- **0 falsos positivos:** El modelo no cometió errores prediciendo que la calidad del aire era buena cuando no lo era.
- **1 falso negativo:** El modelo predijo incorrectamente que la calidad del aire era mala cuando en realidad era buena.
- **72 verdaderos positivos:** El modelo predijo correctamente que la calidad del aire era buena en 72 ocasiones.

```
Evaluación en el conjunto de validación:  
Precisión (Accuracy): 0.9933333333333333  
Precisión: 0.9935897435897436  
Recall: 0.9931506849315068  
F1 Score: 0.9933259176863181  
Matriz de confusión:  
[[77  0]  
 [ 1 72]]
```

Evaluación en el Conjunto de Prueba

Precisión (Accuracy): 0.9867

- Esto indica que el 98.67% de las predicciones en el conjunto de prueba fueron correctas, a pesar de estar ligeramente por debajo de la precisión en el conjunto de validación, sigue siendo un valor muy alto.

Precisión: 0.9872

- La precisión en el conjunto de prueba es del 98.72%, lo que significa que casi todas las veces que el modelo predijo que la calidad del aire era buena, acertó.

Recall: 0.9865

- El recall indica que el modelo identificó correctamente el 98.65% de los casos positivos (buena calidad del aire) en el conjunto de prueba.

F1 Score: 0.9866

- El F1 Score en el conjunto de prueba es del 98.66%, lo que muestra un excelente equilibrio entre la precisión y el recall.

Matriz de Confusión (Prueba):

- [[72, 2], [0, 76]]

La matriz de confusión en el conjunto de prueba indica lo siguiente:

- o **72 verdaderos negativos:** El modelo predijo correctamente que la calidad del aire era mala en 72 ocasiones.
- o **2 falsos positivos:** En 2 ocasiones, el modelo predijo incorrectamente que la calidad del aire era buena cuando no lo era.

- **0 falsos negativos:** No hubo casos en los que el modelo predijera incorrectamente que la calidad del aire era mala cuando en realidad era buena.
- **76 verdaderos positivos:** El modelo predijo correctamente que la calidad del aire era buena en 76 ocasiones.

```
Evaluación en el conjunto de prueba:
Precisión (Accuracy): 0.9866666666666667
Precisión: 0.9871794871794872
Recall: 0.9864864864864865
F1 Score: 0.9866571784380003
Matriz de confusión (Prueba):
[[72  2]
 [ 0 76]]
```

Conclusión:

Este proyecto se centró en la implementación de un modelo de regresión logística para predecir la calidad del aire, abarcando todo el flujo de trabajo desde la carga y preprocesamiento de los datos hasta la evaluación del modelo en conjuntos de validación y prueba, a lo largo del proyecto, se desarrolló un modelo robusto, optimizado mediante técnicas de selección de características y ajuste de hiperparámetros, lo que nos permitió explorar y comprender los mecanismos clave de este tipo de modelos de clasificación.

El modelo mostró un rendimiento sobresaliente tanto en los conjuntos de validación como en los de prueba, con precisiones superiores al 98% y F1 Scores que reflejan un excelente equilibrio entre precisión y recall; esto indica que el modelo fue capaz de capturar patrones significativos en los datos, logrando una separación clara entre los casos de buena y mala calidad del aire. Aunque el rendimiento general fue alto, algunos errores mínimos en la clasificación de falsos positivos y falsos negativos sugieren que hay espacio para mejorar el ajuste del modelo.

La normalización de los datos y la selección de las características más importantes mediante RFE fueron pasos cruciales en el proceso, lo que resalta la importancia de un preprocesamiento adecuado en cualquier proyecto de aprendizaje automático, estos pasos ayudaron a garantizar que el modelo trabajara con datos limpios y escalados, lo que contribuyó a su precisión y consistencia en diferentes escenarios.

Finalmente, la implementación del código para realizar predicciones con nuevos datos, junto con la evaluación detallada del modelo en diferentes conjuntos, demuestra la aplicabilidad y flexibilidad de la solución propuesta. La capacidad de ingresar nuevas características y obtener predicciones rápidas sobre la calidad del aire convierte este proyecto en una herramienta práctica y valiosa para su uso en aplicaciones reales.

En resumen, este proyecto no solo permitió implementar un modelo funcional, sino que también proporcionó una comprensión profunda de los algoritmos de regresión logística y la importancia de cada fase en el flujo de trabajo del aprendizaje automático, los resultados obtenidos son altamente satisfactorios, y con algunos ajustes adicionales, el modelo podría mejorarse aún más para aplicaciones más complejas en la evaluación de la calidad del aire u otras tareas de clasificación.