

Actividad 3.6

DengAI: predicción de la propagación de enfermedades



ÍNDICE

| | |
|---|-----------|
| Verificar versiones..... | 3 |
| Cargar los datos..... | 4 |
| Unir las etiquetas con los datos de entrenamiento..... | 5 |
| Preprocesamiento..... | 5 |
| Entrenar modelos..... | 7 |
| Random Forest..... | 7 |
| XG Boost..... | 8 |
| Regresión Lineal..... | 8 |
| Hiper Parámetros Óptimos con Grid Search CV..... | 9 |
| Hiper Parámetros con Randomized Search CV..... | 10 |
| Alinear las columnas de X_test con X_train..... | 11 |
| Cargar y Guardar Formato de Submission con Predicciones..... | 11 |
| Conclusiones de los Resultados:..... | 12 |
| Resumen..... | 12 |
| REPOSITORIOS..... | 13 |
| BUSCANDO MEJORAS..... | 13 |
| GridSearchCV: Búsqueda exhaustiva sobre un conjunto de parámetros predefinidos... | 15 |
| RandomizedSearchCV: Búsqueda aleatoria de hiperparámetros..... | 15 |
| Evaluación de los resultados..... | 16 |
| Implementación de XG Boost:..... | 16 |
| Implementación con Optuna:..... | 17 |
| pipeline con RandomForest..... | 18 |
| Conclusiones de los nuevos modelos y predicciones:..... | 19 |
| Resultados competición..... | 20 |

Verificar versiones

✓ Iván Falcón Monzón

Actividad 3.6 - DengAI: predicción de la propagación de enfermedades

Archivos de la competición

dengue_features_train.csv → Características de entrenamiento. dengue_labels_train.csv → Etiquetas (casos totales de dengue).

dengue_features_test.csv → Características de prueba (para predicción). submission_format.csv → Formato para subir el archivo de predicciones.

Verificar versiones compatibles con el código:

```
✓ 4 s [1] 1 # IVAN FALCON MONZON
2 import sklearn
3 import numpy as np
4 import xgboost
5
6 print("Scikit-learn version:", sklearn.__version__)
7 print("NumPy version:", np.__version__)
8 print("XGBoost version:", xgboost.__version__)
```

```
Scikit-learn version: 1.3.0
NumPy version: 1.23.5
XGBoost version: 2.0.3
```

Scikit-learn version: 1.3.0

NumPy version: 1.23.5

XGBoost version: 2.0.3

He tenido problemas con los modelos y las versiones, asique si no funciona el código instalar las nuevas versiones.

```
✓ 0 s [2] 1 #!pip uninstall -y scikit-learn numpy
2 #!pip install --no-cache-dir numpy==1.23.5 scikit-learn==1.3.0
3 #!pip install --force-reinstall xgboost==2.0.3
```

Cargar los datos

Yo utilizo repositorio en github:

✓ Cargar los datos

✓
0 s

```
[3] 1 # IVAN FALCON MONZON
    2 import pandas as pd
    3
    4 # Enlaces directos a los archivos CSV en GitHub (debes reemplazarlos con los reales)
    5 github_base = "https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_6_IvanFalconMonzon/main/"
    6
    7 # Cargar datasets desde GitHub
    8 train_features = pd.read_csv(github_base + "dengue_features_train.csv")
    9 train_labels = pd.read_csv(github_base + "dengue_labels_train.csv")
   10 test_features = pd.read_csv(github_base + "dengue_features_test.csv")
   11
   12 # Verificar estructura de los datasets
   13 print(train_features.head())
   14 print(train_labels.head())
   15 print(test_features.head())
```



| | station_max_temp_c | station_min_temp_c | station_precip_mm |
|---|--------------------|--------------------|-------------------|
| 0 | 29.4 | 20.0 | 16.0 |
| 1 | 31.7 | 22.2 | 8.6 |
| 2 | 32.2 | 22.8 | 41.4 |
| 3 | 33.3 | 23.3 | 4.0 |
| 4 | 35.0 | 23.9 | 5.8 |

[5 rows x 24 columns]

| | city | year | weekofyear | total_cases |
|---|------|------|------------|-------------|
| 0 | sj | 1990 | 18 | 4 |
| 1 | sj | 1990 | 19 | 5 |
| 2 | sj | 1990 | 20 | 4 |
| 3 | sj | 1990 | 21 | 3 |
| 4 | sj | 1990 | 22 | 6 |

| | city | year | weekofyear | week_start_date | ndvi_ne | ndvi_nw | ndvi_se \ |
|---|------|------|------------|-----------------|---------|-----------|-----------|
| 0 | sj | 2008 | 18 | 2008-04-29 | -0.0189 | -0.018900 | 0.102729 |
| 1 | sj | 2008 | 19 | 2008-05-06 | -0.0180 | -0.012400 | 0.082043 |
| 2 | sj | 2008 | 20 | 2008-05-13 | -0.0015 | NaN | 0.151083 |
| 3 | sj | 2008 | 21 | 2008-05-20 | NaN | -0.019867 | 0.124329 |
| 4 | sj | 2008 | 22 | 2008-05-27 | 0.0568 | 0.039833 | 0.062267 |

Unir las etiquetas con los datos de entrenamiento

✓ Unir las etiquetas con los datos de entrenamiento

✓
0 s

```
[4] 1 # IVAN FALCON MONZON
2 # Unir las etiquetas con las características (clave: 'city', 'year', 'weekofyear')
3 train_data = train_features.merge(train_labels, on=['city', 'year', 'weekofyear'])
4
5 # Separar características (X) y variable objetivo (y)
6 X = train_data.drop(columns=['city', 'year', 'weekofyear', 'total_cases'])
7 y = train_data['total_cases']
8 X_test = test_features.drop(columns=['city', 'year', 'weekofyear'])
```

Preprocesamiento

✓ Preprocesamiento

1. Rellenar valores nulos (forward fill):

✓
0 s

```
[5] 1 # IVAN FALCON MONZON
2 X = X.fillna(method='ffill')
3 X_test = X_test.fillna(method='ffill')
```



```
<ipython-input-5-a046242418a9>:2: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
X = X.fillna(method='ffill')
<ipython-input-5-a046242418a9>:3: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
X_test = X_test.fillna(method='ffill')
```

2. División de datos en entrenamiento y validación

✓
0 s

```
[6] 1 # IVAN FALCON MONZON
2 from sklearn.model_selection import train_test_split # Importa la función para dividir los datos
3
4 # Divide el conjunto de datos en entrenamiento (80%) y validación (20%)
5 X_train, X_val, y_train, y_val = train_test_split(
6     X, y,          # X: características, y: variable objetivo
7     test_size=0.2, # 20% de los datos se usarán para validación
8     random_state=42 # Fija una semilla para obtener siempre la misma división
9 )
```

Conversión de fechas a valores numéricos

✓
0 s

```
[7] 1 # IVAN FALCON MONZON
2 # Convertir la columna 'week_start_date' a tipo datetime
3 X['week_start_date'] = pd.to_datetime(X['week_start_date'])
4 X_test['week_start_date'] = pd.to_datetime(X_test['week_start_date'])
5
6 # Definir una fecha base para calcular los días transcurridos
7 base_date = pd.to_datetime("1990-01-01")
8
9 # Crear una nueva columna con la cantidad de días desde la fecha base
10 X['days_since_start'] = (X['week_start_date'] - base_date).dt.days
11 X_test['days_since_start'] = (X_test['week_start_date'] - base_date).dt.days
12
13 # Eliminar la columna original de fecha, ya que ahora tenemos una versión numérica
14 X = X.drop(columns=['week_start_date'])
15 X_test = X_test.drop(columns=['week_start_date'])
```

Eliminación segura de la columna week_start_date

✓
0 s

```
[8] 1 # IVAN FALCON MONZON
2 # Eliminar la columna 'week_start_date' si existe, evitando errores si ya fue eliminada
3 X_train = X_train.drop(columns=['week_start_date'], errors='ignore')
4 X_val = X_val.drop(columns=['week_start_date'], errors='ignore')
5 X_test = X_test.drop(columns=['week_start_date'], errors='ignore')
```

Conversión de la fecha 'week_start_date' a formato datetime y extracción de componentes

✓
0 s

```
1 # IVAN FALCON MONZON
2 # Convertir 'week_start_date' a formato datetime y extraer Año, Mes, Día en TODOS los conjuntos
3 for df in [X_train, X_val, X_test]: # Iterar sobre cada uno de los conjuntos de datos (entrenamiento, validación y prueba)
4     if 'week_start_date' in df.columns: # Comprobar si la columna 'week_start_date' existe en el DataFrame
5         # Convertir la columna 'week_start_date' a tipo datetime (para manipulación de fechas)
6         df['week_start_date'] = pd.to_datetime(df['week_start_date'])
7
8         # Extraer el año de la columna 'week_start_date' y guardarlo en una nueva columna 'year'
9         df['year'] = df['week_start_date'].dt.year
10
11         # Extraer el mes de la columna 'week_start_date' y guardarlo en una nueva columna 'month'
12         df['month'] = df['week_start_date'].dt.month
13
14         # Extraer el día de la columna 'week_start_date' y guardarlo en una nueva columna 'day'
15         df['day'] = df['week_start_date'].dt.day
16
17         # Eliminar la columna original 'week_start_date' ya que la información ahora está separada en nuevas columnas
18         df.drop(columns=['week_start_date'], inplace=True)
```

Entrenar modelos

Random Forest

Entrenar modelos

Modelo 1: Random Forest

Verificar si hay valores de tipo string en los datos

```
[10] 1 # IVAN FALCON MONZON
      2 print("Columnas en X_train:", X_train.columns)
      3 print("Columnas en X_val:", X_val.columns)
```

Columnas en X_train: Index(['ndvi_ne', 'ndvi_nw', 'ndvi_se', 'ndvi_sw', 'precipitation_amt_mm', 'reanalysis_air_temp_k', 'reanalysis_avg_temp_k', 'reanalysis_dew_point_temp_k', 'reanalysis_max_air_temp_k', 'reanalysis_min_air_temp_k', 'reanalysis_precip_amt_kg_per_m2', 'reanalysis_relative_humidity_percent', 'reanalysis_sat_precip_amt_mm', 'reanalysis_specific_humidity_g_per_kg', 'reanalysis_tdt_r_k', 'station_avg_temp_c', 'station_diur_temp_rng_c', 'station_max_temp_c', 'station_min_temp_c', 'station_precip_mm'], dtype='object')

Columnas en X_val: Index(['ndvi_ne', 'ndvi_nw', 'ndvi_se', 'ndvi_sw', 'precipitation_amt_mm', 'reanalysis_air_temp_k', 'reanalysis_avg_temp_k', 'reanalysis_dew_point_temp_k', 'reanalysis_max_air_temp_k', 'reanalysis_min_air_temp_k', 'reanalysis_precip_amt_kg_per_m2', 'reanalysis_relative_humidity_percent', 'reanalysis_sat_precip_amt_mm', 'reanalysis_specific_humidity_g_per_kg', 'reanalysis_tdt_r_k', 'station_avg_temp_c', 'station_diur_temp_rng_c', 'station_max_temp_c', 'station_min_temp_c', 'station_precip_mm'], dtype='object')

Entrenamiento y Evaluación de un Modelo Random Forest

```
[11] 1 # IVAN FALCON MONZON
      2
      3 # Importar el modelo RandomForestRegressor de la librería sklearn
      4 from sklearn.ensemble import RandomForestRegressor
      5 # Importar la función mean_absolute_error para calcular el error absoluto medio
      6 from sklearn.metrics import mean_absolute_error
      7
      8 # Crear una instancia del modelo RandomForestRegressor con 100 árboles (n_estimators) y una semilla para la aleatoriedad (random_state)
      9 rf = RandomForestRegressor(n_estimators=100, random_state=42)
     10 # Ajustar el modelo a los datos de entrenamiento (X_train y y_train)
     11 rf.fit(X_train, y_train)
     12
     13 # Realizar predicciones sobre el conjunto de validación (X_val)
     14 y_pred_rf = rf.predict(X_val)
     15
     16 # Calcular el error absoluto medio (MAE) comparando las predicciones con los valores reales del conjunto de validación (y_val)
     17 mae_rf = mean_absolute_error(y_val, y_pred_rf)
     18
     19 # Mostrar el resultado del MAE para el modelo Random Forest
     20 print(f"MAE - Random Forest: {mae_rf}")
```

MAE - Random Forest: 19.546690639269407

XG Boost

Entrenamiento y Evaluación de un Modelo XGBoost

```
[12] 1 # IVAN FALCON MONZON
      2
      3 # Importar el modelo XGBRegressor de la librería xgboost
      4 from xgboost import XGBRegressor
      5
      6 # Crear una instancia del modelo XGBRegressor con 100 estimadores, tasa de aprendizaje de 0.1 y semilla para la aleatoriedad
      7 xgb = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
      8
      9 # Ajustar el modelo a los datos de entrenamiento (X_train y y_train)
     10 xgb.fit(X_train, y_train)
     11
     12 # Realizar predicciones sobre el conjunto de validación (X_val)
     13 y_pred_xgb = xgb.predict(X_val)
     14
     15 # Calcular el error absoluto medio (MAE) comparando las predicciones con los valores reales del conjunto de validación (y_val)
     16 mae_xgb = mean_absolute_error(y_val, y_pred_xgb)
     17
     18 # Mostrar el resultado del MAE para el modelo XGBoost
     19 print(f"MAE - XGBoost: {mae_xgb}")
```

MAE - XGBoost: 19.133190783123446

Regresión Lineal

Entrenamiento y Evaluación de un Modelo de Regresión Lineal

```
[13] 1 # IVAN FALCON MONZON
      2
      3 # Importar el modelo LinearRegression de la librería sklearn
      4 from sklearn.linear_model import LinearRegression
      5
      6 # Crear una instancia del modelo LinearRegression
      7 lr = LinearRegression()
      8
      9 # Ajustar el modelo a los datos de entrenamiento (X_train y y_train)
     10 lr.fit(X_train, y_train)
     11
     12 # Realizar predicciones sobre el conjunto de validación (X_val)
     13 y_pred_lr = lr.predict(X_val)
     14
     15 # Calcular el error absoluto medio (MAE) comparando las predicciones con los valores reales del conjunto de validación (y_val)
     16 mae_lr = mean_absolute_error(y_val, y_pred_lr)
     17
     18 # Mostrar el resultado del MAE para el modelo de regresión lineal
     19 print(f"MAE - Regresión Lineal: {mae_lr}")
```

MAE - Regresión Lineal: 25.343064369691845

Hiper Parámetros Óptimos con Grid Search CV

Búsqueda de Hiperparámetros Óptimos con GridSearchCV

```
1 # IVAN FALCON MONZON
2
3 # Importar GridSearchCV desde la librería sklearn.model_selection para la búsqueda de hiperparámetros
4 from sklearn.model_selection import GridSearchCV
5
6 # Definir la cuadrícula de parámetros a evaluar (n_estimators, max_depth, min_samples_split)
7 param_grid = {
8     'n_estimators': [50, 100, 200],
9     'max_depth': [10, 20, None],
10    'min_samples_split': [2, 5, 10]
11 }
12
13 # Crear una instancia de GridSearchCV, aplicando RandomForestRegressor y los parámetros definidos, con validación cruzada de 3 pliegues (cv=3)
14 # Se utiliza 'neg_mean_absolute_error' como la métrica para la evaluación
15 grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=3, scoring='neg_mean_absolute_error')
16
17 # Ajustar el GridSearchCV a los datos de entrenamiento (X_train, y_train)
18 grid_search.fit(X_train, y_train)
19
20 # Imprimir los mejores parámetros encontrados
21 print("Mejores parámetros:", grid_search.best_params_)
22
23 # Imprimir el mejor MAE (negativo, por eso se multiplica por -1)
24 print("Mejor MAE:", -grid_search.best_score_)
```

➡ Mejores parámetros: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
Mejor MAE: 16.651553436750692

Hiper Parámetros con Randomized Search CV

Búsqueda Aleatoria de Hiperparámetros con RandomizedSearchCV

✓
33 s

```
[15] 1 # IVAN FALCON MONZON
2
3 # Importar RandomizedSearchCV desde la librería sklearn.model_selection para la búsqueda aleatoria de hiperparámetros
4 from sklearn.model_selection import RandomizedSearchCV
5 # Importar XGBRegressor de la librería xgboost
6 from xgboost import XGBRegressor
7
8 # Definir la distribución de parámetros a explorar (n_estimators, learning_rate, max_depth)
9 param_dist = {
10     'n_estimators': [50, 100, 200],
11     'learning_rate': [0.01, 0.1, 0.2],
12     'max_depth': [3, 6, 10]
13 }
14
15 # Inicializar el modelo XGBRegressor antes de pasarlo a RandomizedSearchCV
16 xgb_regressor = XGBRegressor()
17
18 # Crear una instancia de RandomizedSearchCV, con 10 iteraciones (n_iter=10), validación cruzada de 3 pliegues (cv=3)
19 # Se utiliza 'neg_mean_absolute_error' como la métrica de evaluación
20 random_search = RandomizedSearchCV(
21     xgb_regressor, # Usar la instancia de XGBRegressor inicializada
22     param_dist,
23     n_iter=10, # Número de combinaciones aleatorias a probar
24     cv=3, # Número de pliegues en la validación cruzada
25     scoring='neg_mean_absolute_error', # Usar el error absoluto medio negativo como la métrica de evaluación
26     random_state=42, # Semilla para la aleatoriedad
27     n_jobs=1 # Usar un solo hilo para evitar problemas con múltiples hilos
28 )
29
30 # Ajustar el RandomizedSearchCV a los datos de entrenamiento (X_train, y_train)
31 random_search.fit(X_train, y_train)
32
33 # Imprimir los mejores parámetros encontrados durante la búsqueda
34 print("Mejores parámetros:", random_search.best_params_)
35
36 # Imprimir el mejor MAE (negativo, por eso se multiplica por -1)
37 print("Mejor MAE:", -random_search.best_score_)
```



Mejores parámetros: {'n_estimators': 50, 'max_depth': 3, 'learning_rate': 0.1}
Mejor MAE: 16.614883451271304

Alinear las columnas de X_test con X_train

Alinear las columnas de X_test con X_train

```
[16] 1 # IVAN FALCON MONZON
      2
      3 # Asegurar que las columnas de X_test estén alineadas con las columnas de X_train
      4 # Esto es útil si ha habido cambios en las columnas de X_train (por ejemplo, después de la selección de características) y queremos que X_test tenga las mismas columnas
      5 X_test = X_test[X_train.columns]
```

Cargar y Guardar Formato de Submission con Predicciones

Cargar y Guardar Formato de Submission con Predicciones

```
[22] 1 import pandas as pd
      2 from google.colab import files
      3
      4 # Enlace directo al archivo en GitHub (verifica que sea el correcto)
      5 github_base = "https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_6_IvanFalconMonzon/main/"
      6
      7 # Cargar el formato de submission desde GitHub (archivo CSV que contiene la estructura esperada)
      8 submission = pd.read_csv(github_base + "submission_format.csv")
      9
      10 # Asegúrate de tener tus datos de predicciones listos (por ejemplo, predictions)
      11 # Aquí asumiré que 'predictions' es la variable con las predicciones generadas
      12 submission['total_cases'] = predictions.round().astype(int)
      13
      14 # Guardar el archivo CSV en Colab
      15 submission.to_csv("submission.csv", index=False)
      16
      17 # Descargar el archivo automáticamente en la carpeta de "Descargas" de tu equipo
      18 files.download("submission.csv")
      19
      20 # Imprimir mensaje de confirmación
      21 print("El archivo submission.csv se ha descargado automáticamente.")
```



El archivo submission.csv se ha descargado automáticamente.

Conclusiones de los Resultados:

▼ Conclusiones de los Resultados:

1. Comparación de Modelos Básicos:

- **Random Forest:** El modelo de Random Forest tiene un MAE de 19.55, lo que indica un rendimiento razonable pero no el mejor.
- **XGBoost:** El modelo de XGBoost tiene un MAE de 19.13, lo que lo coloca ligeramente por encima del Random Forest. Esto sugiere que XGBoost podría estar manejando mejor las características y las interacciones en los datos que Random Forest.
- **Regresión Lineal:** El modelo de Regresión Lineal tiene un MAE de 25.34, lo que indica que este modelo es menos efectivo en este caso. La regresión lineal es probablemente demasiado simple para capturar las relaciones complejas en los datos.

2. Optimización de Hiperparámetros:

- **GridSearchCV (Búsqueda Exhaustiva):** La búsqueda de los mejores hiperparámetros con GridSearchCV resultó en un MAE de 16.65, lo que es una mejora notable sobre los modelos básicos. Los mejores parámetros fueron:
 - `max_depth = 10`
 - `min_samples_split = 2`
 - `n_estimators = 200` Esto muestra que la optimización de los parámetros tiene un impacto positivo en el rendimiento del modelo.
- **RandomizedSearchCV (Búsqueda Aleatoria):** La búsqueda de hiperparámetros con RandomizedSearchCV dio como resultado un MAE de 16.61, que es aún más bajo que el de GridSearchCV. Los mejores parámetros fueron:
 - `n_estimators = 50`
 - `max_depth = 3`
 - `learning_rate = 0.1` Este método también encontró un conjunto de parámetros efectivos, aunque no fue tan exhaustivo como GridSearchCV.

Resumen

Resumen:

- XGBoost y Random Forest son los modelos más prometedores, con XGBoost teniendo un rendimiento ligeramente superior.
- La Regresión Lineal no es tan efectiva en este caso, ya que su MAE es más alto.
- Las búsquedas de hiperparámetros (tanto con GridSearchCV como RandomizedSearchCV) mejoraron significativamente el rendimiento, con RandomizedSearchCV mostrando el mejor MAE.
- Las optimizaciones sugieren que el modelo final con los parámetros óptimos puede mejorar considerablemente el rendimiento en comparación con los modelos básicos.

REPOSITORIOS

Google Colab: <https://colab.research.google.com/drive/1Jy5ZJmDnXHav2Co6hupYFYCffjD8T3tF?usp=sharing>

Github: https://github.com/IvanFalconMonzon/SNS_ACT3_6_IvanFalconMonzon.git

BUSCANDO MEJORAS

```
# IVAN FALCON MONZON
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer

# Cargar datasets
github_base
"https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_6_IvanFalconMonzon/main/"
train_features = pd.read_csv(github_base + "dengue_features_train.csv")
train_labels = pd.read_csv(github_base + "dengue_labels_train.csv")
test_features = pd.read_csv(github_base + "dengue_features_test.csv")

# Identificar columnas no numéricas
non_numeric_columns = train_features.select_dtypes(include=['object']).columns
print(f"Columnas con valores no numéricos: {non_numeric_columns}")

# Transformar 'week_start_date' a características numéricas
train_features['week_start_date'] = pd.to_datetime(train_features['week_start_date'])
train_features['year'] = train_features['week_start_date'].dt.year
train_features['month'] = train_features['week_start_date'].dt.month
train_features['day'] = train_features['week_start_date'].dt.day
train_features.drop(columns=['week_start_date'], inplace=True)

# Eliminar la columna 'city' o aplicar OneHotEncoding
train_features = pd.get_dummies(train_features, columns=['city'], drop_first=True)

# Limpiar 'y_train' eliminando o corrigiendo valores no numéricos
train_labels = pd.to_numeric(train_labels['total_cases'], errors='coerce') # Convertir a
NaN los valores no numéricos
train_labels = train_labels.dropna() # Eliminar filas con valores NaN

# Asegurarnos de que train_features y train_labels tengan el mismo número de muestras
train_features = train_features.iloc[:len(train_labels), :]
```

```

# Imputar los valores NaN con la media de cada columna
imputer = SimpleImputer(strategy='mean')
train_features_imputed = imputer.fit_transform(train_features)

# Dividir en datos de entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(train_features_imputed, train_labels,
test_size=0.2, random_state=42)

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Entrenamiento y predicción con RandomForestRegressor
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(X_train_scaled, y_train)
y_pred_rf = model_rf.predict(X_val_scaled)

# Evaluar el modelo
mae_rf = mean_absolute_error(y_val, y_pred_rf)
print(f"MAE - Random Forest después de limpiar y transformar datos: {mae_rf}")

```

Columnas con valores no numéricos: Index(['city', 'week_start_date'], dtype='object')

MAE - Random Forest después de limpiar y transformar datos: 14.61359589041096

GridSearchCV: Búsqueda exhaustiva sobre un conjunto de parámetros predefinidos

GridSearchCV: Búsqueda exhaustiva sobre un conjunto de parámetros predefinidos

```
1 # IVAN FALCON MONZON
2 from sklearn.model_selection import GridSearchCV
3 from sklearn.ensemble import RandomForestRegressor
4
5 # Definir el rango de hiperparámetros que quieres probar
6 param_grid = {
7     'n_estimators': [50, 100, 200],      # Número de árboles en el bosque
8     'max_depth': [10, 20, None],         # Profundidad máxima de los árboles
9     'min_samples_split': [2, 5, 10],     # Mínimo número de muestras requeridas para dividir un nodo
10    'min_samples_leaf': [1, 2, 4],        # Mínimo número de muestras en una hoja
11    'bootstrap': [True, False],           # Si se debe usar muestreo con reemplazo
12 }
13
14 # Crear el modelo de RandomForest
15 model_rf = RandomForestRegressor(random_state=42)
16
17 # Realizar la búsqueda de hiperparámetros
18 grid_search = GridSearchCV(estimator=model_rf, param_grid=param_grid, cv=3, scoring='neg_mean_absolute_error', n_jobs=-1)
19 grid_search.fit(X_train_scaled, y_train)
20
21 # Mostrar los mejores parámetros y el mejor MAE encontrado
22 print("Mejores parámetros:", grid_search.best_params_)
23 print("Mejor MAE:", -grid_search.best_score_)
```

➔ Mejores parámetros: {'bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Mejor MAE: 13.499927798467558

RandomizedSearchCV: Búsqueda aleatoria de hiperparámetros

RandomizedSearchCV: Búsqueda aleatoria de hiperparámetros

```
[21] 1 # IVAN FALCON MONZON
2 from sklearn.model_selection import RandomizedSearchCV
3 from sklearn.ensemble import RandomForestRegressor
4 import numpy as np
5
6 # Definir el rango de hiperparámetros que quieres probar
7 param_dist = {
8     'n_estimators': [50, 100, 200, 300], # Número de árboles en el bosque
9     'max_depth': [10, 20, 30, None],     # Profundidad máxima de los árboles
10    'min_samples_split': [2, 5, 10],      # Mínimo número de muestras requeridas para dividir un nodo
11    'min_samples_leaf': [1, 2, 4],        # Mínimo número de muestras en una hoja
12    'bootstrap': [True, False],           # Si se debe usar muestreo con reemplazo
13 }
14
15 # Crear el modelo de RandomForest
16 model_rf = RandomForestRegressor(random_state=42)
17
18 # Realizar la búsqueda de hiperparámetros con una cantidad limitada de combinaciones aleatorias
19 random_search = RandomizedSearchCV(estimator=model_rf, param_distributions=param_dist, n_iter=10, cv=3,
20                                   scoring='neg_mean_absolute_error', random_state=42, n_jobs=-1)
21 random_search.fit(X_train_scaled, y_train)
22
23 # Mostrar los mejores parámetros y el mejor MAE encontrado
24 print("Mejores parámetros:", random_search.best_params_)
25 print("Mejor MAE:", -random_search.best_score_)
```

➔ Mejores parámetros: {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 20, 'bootstrap': True}
Mejor MAE: 13.862199448481727

Evaluación de los resultados

Evaluación de los resultados

```
[22] 1 # IVAN FALCON MONZON
      2 # Usar los mejores parámetros encontrados en GridSearchCV o RandomizedSearchCV
      3 best_model = random_search.best_estimator_
      4
      5 # Entrenamiento y predicción con el mejor modelo
      6 best_model.fit(X_train_scaled, y_train)
      7 y_pred_best = best_model.predict(X_val_scaled)
      8
      9 # Evaluación del modelo
     10 mae_best = mean_absolute_error(y_val, y_pred_best)
     11 print(f"Mejor MAE después de optimizar hiperparámetros: {mae_best}")
```

⇒ Mejor MAE después de optimizar hiperparámetros: 14.751873705416706

Implementación de XG Boost:

Implementación de XGBoost:

```
[22] 1 # IVAN FALCON MONZON
      2 import xgboost as xgb
      3 from sklearn.metrics import mean_absolute_error
      4
      5 # Entrenamiento de XGBoost
      6 model_xgb = xgb.XGBRegressor(n_estimators=200, max_depth=10, learning_rate=0.1, random_state=42)
      7 model_xgb.fit(X_train_scaled, y_train)
      8
      9 # Predicciones
     10 y_pred_xgb = model_xgb.predict(X_val_scaled)
     11
     12 # Evaluación
     13 mae_xgb = mean_absolute_error(y_val, y_pred_xgb)
     14 print(f"MAE - XGBoost: ", mae_xgb)
```

⇒ MAE - XGBoost: 14.37872043448462

Implementación con Optuna:

Implementación con Optuna:

Optuna es una biblioteca de optimización automática de hiperparámetros, utilizada para mejorar el rendimiento de los modelos de machine learning. Funciona de manera eficiente utilizando algoritmos avanzados de optimización como el algoritmo de búsqueda de bayesiana para explorar y encontrar los mejores valores de los hiperparámetros.

```
[ ] 1 # IVAN FALCON MONZON
    2 # Instalar optuna
    3 !pip install optuna

[24] 1 # IVAN FALCON MONZON
    2 import optuna
    3 from sklearn.ensemble import RandomForestRegressor
    4 from sklearn.metrics import mean_absolute_error
    5
    6 # Definimos la función de objetivo para Optuna
    7 def objective(trial):
    8     model = RandomForestRegressor(
    9         n_estimators=trial.suggest_int('n_estimators', 50, 200),
   10         max_depth=trial.suggest_int('max_depth', 5, 20),
   11         min_samples_split=trial.suggest_int('min_samples_split', 2, 10),
   12         min_samples_leaf=trial.suggest_int('min_samples_leaf', 1, 5),
   13         bootstrap=trial.suggest_categorical('bootstrap', (True, False)),
   14         random_state=42
   15     )
   16     model.fit(X_train_scaled, y_train)
   17     y_pred = model.predict(X_val_scaled)
   18     return mean_absolute_error(y_val, y_pred)
   19
   20 # Crear un estudio y optimizar
   21 study = optuna.create_study(direction="minimize")
   22 study.optimize(objective, n_trials=50)
   23
   24 # Imprimir los mejores parámetros y el MAE
   25 print(f"Mejores parámetros: {study.best_params}")
   26 print(f"Mejor MAE: {study.best_value}")

[ 2025-02-14 10:58:22,370] A new study created in memory with name: no-name-d0c93ec4-c51c-4632-8261-af9bb45a4bfd
[ 2025-02-14 10:58:25,057] Trial 0 finished with value: 15.805904332777192 and parameters: {'n_estimators': 96, 'max_depth': 18, 'min_samples_split': 3, 'min_s
[ 2025-02-14 10:58:27,180] Trial 1 finished with value: 15.011170666739462 and parameters: {'n_estimators': 128, 'max_depth': 14, 'min_samples_split': 6, 'min
```

Mejores parámetros: {'n_estimators': 169, 'max_depth': 13, 'min_samples_split': 4, 'min_samples_leaf': 2, 'bootstrap': True}
Mejor MAE: 14.375869807302005

pipeline con RandomForest

pipeline con RandomForest

```
1 # IVAN FALCON MONZON
2 from sklearn.pipeline import Pipeline
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.impute import SimpleImputer
6
7 # Pipeline con imputación, escalado y RandomForest
8 pipeline = Pipeline([
9     ('imputer', SimpleImputer(strategy='mean')),
10    ('scaler', StandardScaler()),
11    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
12 ])
13
14 # Entrenamiento y evaluación
15 pipeline.fit(X_train, y_train)
16 y_pred = pipeline.predict(X_val)
17 mae = mean_absolute_error(y_val, y_pred)
18 print("MAE con pipeline: ", mae)
```

MAE con pipeline: 14.61359589041096

Descargar fichero

```
[ ] 1 # IVAN FALCON MONZON
2 from google.colab import files
3
4 # Asegúrate de que el archivo con el nombre correcto esté en el entorno de Colab
5 files.download('submission.csv') # Cambia el nombre si es necesario
```

Conclusiones de los nuevos modelos y predicciones:

✓ Conclusiones de los nuevos modelos y predicciones:

1. Random Forest (después de limpiar y transformar datos):

- El MAE es 14.61, lo que indica un desempeño moderado tras la limpieza y transformación de los datos.

2. GridSearchCV:

- Mejores parámetros: {'bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}.
- Mejor MAE: 13.50. Este enfoque de búsqueda exhaustiva mejora el modelo, logrando un MAE más bajo, indicando una optimización efectiva de los hiperparámetros.

3. RandomizedSearchCV:

- Mejores parámetros: {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 20, 'bootstrap': True}.
- Mejor MAE: 13.86. La búsqueda aleatoria también mostró buenos resultados, pero el MAE es ligeramente peor que el obtenido con GridSearchCV.

4. Evaluación de los resultados:

- Después de la optimización de los hiperparámetros, el MAE sube a 14.75, lo que sugiere que aunque hubo mejoras, los resultados no son consistentemente mejores que los obtenidos por otros enfoques.

5. XGBoost:

- MAE: 14.38. Aunque XGBoost es una técnica robusta, el desempeño es ligeramente mejor que el Random Forest estándar pero aún no mejora significativamente los mejores resultados previos.

6. Optuna:

- Mejores parámetros: {'n_estimators': 169, 'max_depth': 13, 'min_samples_split': 4, 'min_samples_leaf': 2, 'bootstrap': True}.
- Mejor MAE: 14.38. Optuna mostró una ligera mejora respecto al Random Forest y XGBoost, pero no superó los mejores resultados de GridSearchCV.

7. Pipeline con RandomForest:

- MAE: 14.61, similar al rendimiento inicial sin pipeline, lo que indica que no se logró una mejora significativa con esta implementación.

Resumen:

En general, el enfoque de GridSearchCV ha mostrado el mejor desempeño con el menor MAE (13.50), mientras que el uso de XGBoost y Optuna también ha sido prometedor, aunque sin superar el rendimiento de GridSearchCV. La optimización de los hiperparámetros es crucial para mejorar el desempeño del modelo, pero la diferencia entre los enfoques no ha sido sustancial.

Resultados competición

Best score

28.2163

Current rank

#5149





Submissions used

3 of 3

No submissions remaining

You have **0 of 3** submissions left today. Your next submission can be on Feb. 15, 2025 UTC.

Your submissions

| Public score | Who | Details |
|--------------|--|--------------------------|
| 28.9351 |  IvanFalconMonzon | id-278922 · 3d 16h ago |
| 28.8125 |  IvanFalconMonzon | id-279064 · 1h 27min ago |
| 29.2308 |  IvanFalconMonzon | id-279067 · 5min ago |
| 28.2163 |  IvanFalconMonzon | id-279068 · 0min ago |