SCRIPT PREDICCION

June 8, 2023

1 Modelo predictivo empleado en el Datathon

El siguiente modelo fue empleado en el Datathon Cajamar UniversityHack. Obtuvo el cuarto mejor resultado de la Universidad Complutense de Madrid entre más de 50 equipos. La justificación del modelo puede encontrarse en el pdf 'Presentacion', donde bajo argumentos científicos se justifica su uso. Dado que se trata de un modelo sencillo y sin algoritmos complejos, se presentarán posteriormente formas más sofisticadas de abordar el problema mediante modelos basados en XGBoost o redes neuronales.

[1]:
```
# Comenzamos importando las librerías necesarias.
import pandas as pd
import numpy as np
import math
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
#Recuperamos el dataset original.
df = pd.read_csv('UH_2023_TRAIN.txt', delimiter = "|")
df22 = df[df['CAMPAÑA'] == 22]
```

[3]:
```
#Cargamos las distancias meteorológicas entre campañas y generamos las␣
 ↪ponderaciones
import pickle
with open('distances_invertido.pickle', 'rb') as handle:
    distancias = pickle.load(handle)
pesos_temporales = np.array(list((reversed([1 / pow(math.e, i) for i in␣
 ↪range(1, 7)]))))
pesos_temporales_11 = np.array(list((reversed([1 / pow(math.e, i) for i in␣
 ↪range(1, 6)]))))
ponderaciones = distancias
for key, value in distancias.items():
    dict_vec = np.array([1/x for x in list(value.values())])
    result = dict_vec * pesos_temporales
    result = result / result.sum()
    ponderaciones[key] = result
```

[4]:
```
# Genereamos un dataframe que muestre la evolución de la producción por finca,␣
 ↪variedad y modo.
df_group=df.groupby(['ID_FINCA', 'VARIEDAD', 'MODO', 'CAMPAÑA']).
 ↪agg({'PRODUCCION':'sum'})
df_group=df_group.unstack().transpose()
df_group
```

[4]:

	ID_FINCA	200	439		447			\
	VARIEDAD	59	9	52	17		40	
	MODO	1	2	2	1	2	2	
	CAMPAÑA							
PRODUCCION	14	1900.000	NaN	2215.2	1824.700	NaN	NaN	
	15	778.104	NaN	3208.4	NaN	3242.106	NaN	
	16	1636.200	NaN	6354.4	864.108	1660.176	NaN	
	17	829.008	NaN	NaN	NaN	1336.986	NaN	
	18	607.212	NaN	NaN	NaN	NaN	NaN	
	19	392.688	NaN	NaN	NaN	NaN	NaN	
	20	545.400	NaN	NaN	NaN	NaN	2828.54	
	21	NaN	1901.402	NaN	NaN	NaN	2037.34	
	22	NaN	0.000	NaN	NaN	NaN	0.00	

	ID_FINCA	523		528	702	…	99033	99108	\
	VARIEDAD	32	59	59	59	…	81	52	
	MODO	2	1	1	2	…	2	2	
	CAMPAÑA					…			
PRODUCCION	14	NaN	2290.4	22780.0	NaN	…	2284.2	4520.0	
	15	NaN	NaN	NaN	NaN	…	NaN	11900.0	
	16	NaN	NaN	NaN	NaN	…	NaN	7510.0	
	17	3732.000	NaN	NaN	NaN	…	NaN	5300.0	
	18	2836.074	NaN	NaN	NaN	…	NaN	5750.0	
	19	1225.824	NaN	NaN	NaN	…	NaN	3300.0	
	20	947.844	NaN	NaN	NaN	…	NaN	6140.0	
	21	745.122	NaN	NaN	NaN	…	NaN	4490.0	
	22	0.000	NaN	NaN	0.0	…	NaN	0.0	

	ID_FINCA	99146		99282	99377		99693	99793	\
	VARIEDAD	17		59	52		81	52	
	MODO	1	2	2	1	2	1	2	
	CAMPAÑA								
PRODUCCION	14	NaN	NaN	6630.663	NaN	NaN	16856.590	NaN	
	15	6480.0	NaN	8000.800	NaN	2280.0	14480.844	NaN	
	16	4080.0	NaN	9230.000	560.0	990.0	15931.125	NaN	
	17	6060.0	NaN	5840.000	NaN	NaN	20130.201	NaN	
	18	NaN	3700.0	9070.000	NaN	2160.0	17597.034	NaN	
	19	NaN	3380.0	7380.000	NaN	1840.0	18405.387	NaN	
	20	NaN	3300.0	6710.000	NaN	2300.0	26876.300	NaN	
	21	NaN	4730.0	8460.000	NaN	2460.0	35418.700	NaN	

```
            22          NaN     0.0     0.000    NaN     0.0      0.000   0.0
```

```
ID_FINCA
VARIEDAD             87
MODO                  2
           CAMPAÑA
PRODUCCION 14        NaN
           15        NaN
           16        NaN
           17        NaN
           18        NaN
           19        NaN
           20        NaN
           21        NaN
           22        0.0

[9 rows x 1946 columns]
```

[5]:
```
# Este código devuelve la esperanza de la producción en las campañas 20 y 21␣
 ↪para una finca que no
# ha producido antes o ha producido algún año suelto. Al predecir habrá fincas␣
 ↪cuya producción no
# podamos estimar en base a producciones anteriores ni a variables␣
 ↪meteorológicas. Queremos ver
# si imputar por la media resulta adecuado.
df_group_trans=df_group.transpose()
df_group_trans
lista = list(df_group_trans[('PRODUCCION', 20)][(df_group_trans.loc[:
 ↪,('PRODUCCION',14)].isna()) & (df_group_trans.loc[:,('PRODUCCION',18)].
 ↪isna()) & (df_group_trans.loc[:,('PRODUCCION',19)].isna()) & (df_group_trans.
 ↪loc[:,('PRODUCCION',20)] > 0)])
lista_21 = list(df_group_trans[('PRODUCCION', 21)][(df_group_trans.loc[:
 ↪,('PRODUCCION',14)].isna()) & (df_group_trans.loc[:,('PRODUCCION',18)].
 ↪isna()) & (df_group_trans.loc[:,('PRODUCCION',20)].isna()) & (df_group_trans.
 ↪loc[:,('PRODUCCION',21)] > 0)])
lista.extend(lista_21)
vector = np.array(lista)
vector.mean()
```

[5]:
```
6311.4863426966285
```

[6]:
```
# Vamos a mostrar las campañas que comenzaron a producir cierta variedad en el␣
 ↪año 2020. Existen
# fincas con valores cercanos a cero y fincas con producciones superiores a la␣
 ↪esperanza de
# producción calculada en la celda anterior. Por ello imputar todas estas␣
 ↪producciones por ese
```

```
# valor no parece óptimo. Hemos comprobado que las fincas que inician con una␣
 ↪producción más alta
# cuentan con niveles de producción más elevados en las demás variedades. Así,␣
 ↪una finca con una
# producción media por variedad de 25.000 que inicia la producción en una nueva␣
 ↪variedad suele
# producir de inicio una cantidad muy superior a la producida por otra finca␣
 ↪con una producción
# media por variedad de 1.000 que igualmente comienza a producir una nueva␣
 ↪variedad. Este hecho se
# tendrá en cuenta en la función definida a continuación. Por lo general, la␣
 ↪esperanza de producción
# en la nueva variedad suele ser ligeramente inferior (alrededor del 75%) a la␣
 ↪producción media de
# la finca por variedad. La variabilidad aumenta a medida que el tamaño de la␣
 ↪finca es mayor, es
# decir, fincas con baja produccion no suelen producir cantidades grandes en␣
 ↪nuevas variedades
# pero fincas con alta producción sí que producen en ocasiones cantidades bajas␣
 ↪en nuevas variedades.
df_group_trans[(df_group_trans.loc[:,('PRODUCCION',14)].isna()) &␣
 ↪(df_group_trans.loc[:,('PRODUCCION',18)].isna()) & (df_group_trans.loc[:
 ↪,('PRODUCCION',19)].isna()) & (df_group_trans.loc[:,('PRODUCCION',20)] > 0)].
 ↪head(30)
```

[6]:

			PRODUCCION						
CAMPAÑA			14	15	16	17	18	19	20
ID_FINCA	VARIEDAD	MODO							
447	40	2	NaN	NaN	NaN	NaN	NaN	NaN	2828.540
3014	17	1	NaN	820.0	830.0	NaN	NaN	NaN	86.000
6454	94	2	NaN	NaN	NaN	NaN	NaN	NaN	6890.000
6950	23	2	NaN	NaN	NaN	NaN	NaN	NaN	9010.000
8623	26	2	NaN	NaN	NaN	NaN	NaN	NaN	1853.500
9270	15	2	NaN	NaN	NaN	NaN	NaN	NaN	16729.768
10925	9	2	NaN	NaN	NaN	NaN	NaN	NaN	18550.000
11220	87	2	NaN	NaN	NaN	NaN	NaN	NaN	14120.000
11587	9	2	NaN	NaN	NaN	NaN	NaN	NaN	7700.000
13333	94	2	NaN	NaN	NaN	NaN	NaN	NaN	9800.000
13871	32	2	NaN	NaN	NaN	NaN	NaN	NaN	9574.448
15071	81	2	NaN	NaN	NaN	NaN	NaN	NaN	5280.000
16921	40	2	NaN	NaN	NaN	NaN	NaN	NaN	18070.000
17378	17	1	NaN	NaN	NaN	NaN	NaN	NaN	216.806
17818	17	1	NaN	NaN	NaN	NaN	NaN	NaN	180.600
18139	52	2	NaN	NaN	NaN	NaN	NaN	NaN	5330.000
18318	81	2	NaN	NaN	NaN	NaN	NaN	NaN	1820.000
19272	81	1	NaN	NaN	NaN	NaN	NaN	NaN	2071.000

20082	15	2	NaN	NaN	NaN	NaN	NaN	NaN	5658.887
20693	15	2	NaN	NaN	NaN	NaN	NaN	NaN	31384.800
22639	40	2	NaN	NaN	NaN	NaN	NaN	NaN	2456.160
24065	94	2	NaN	NaN	NaN	NaN	NaN	NaN	12650.000
24195	87	2	NaN	NaN	NaN	NaN	NaN	NaN	3260.000
25104	23	2	NaN	NaN	NaN	NaN	NaN	NaN	15066.815
26831	40	2	NaN	NaN	NaN	NaN	NaN	NaN	1031.745
29262	23	2	NaN	NaN	NaN	NaN	NaN	NaN	4864.188
31836	87	2	NaN	NaN	NaN	NaN	NaN	NaN	7590.000
32795	26	2	NaN	NaN	NaN	NaN	NaN	NaN	7020.000
33453	32	2	NaN	NaN	NaN	NaN	NaN	NaN	4888.830
34407	92	1	NaN	NaN	NaN	NaN	NaN	NaN	936.000

CAMPAÑA			21	22
ID_FINCA	VARIEDAD	MODO		
447	40	2	2037.340	0.0
3014	17	1	110.000	0.0
6454	94	2	28560.000	0.0
6950	23	2	NaN	NaN
8623	26	2	4344.631	0.0
9270	15	2	23860.724	0.0
10925	9	2	NaN	NaN
11220	87	2	20440.000	0.0
11587	9	2	11640.000	0.0
13333	94	2	8140.000	0.0
13871	32	2	10038.960	0.0
15071	81	2	NaN	0.0
16921	40	2	18460.000	0.0
17378	17	1	277.310	0.0
17818	17	1	231.000	0.0
18139	52	2	7450.000	0.0
18318	81	2	8031.300	0.0
19272	81	1	2981.000	0.0
20082	15	2	5197.600	0.0
20693	15	2	18177.030	0.0
22639	40	2	1710.880	0.0
24065	94	2	10200.000	0.0
24195	87	2	7480.000	0.0
25104	23	2	15687.576	0.0
26831	40	2	743.145	0.0
29262	23	2	11121.936	0.0
31836	87	2	8654.500	0.0
32795	26	2	14930.000	0.0
33453	32	2	12048.453	0.0
34407	92	1	594.000	0.0

[7]:
```
# Esta función da la predicción de producción para la campaña 22 de una finca,␣
 ↪variedad y modo
# concretos. Utiliza las ponderaciones de producción de campañas anteriores␣
 ↪dadas en las variables
# input campañas y pesos. Estas ponderaciones se calculan de manera externa en␣
 ↪base a las similitudes
# meteorológicas de las diferentes campañas teniendo en cuenta la estación a la␣
 ↪que pertenece cada
# finca. Se otorga además un mayor peso a las campañas más próximas en el␣
 ↪tiempo a la campaña 22
#para ajustar la predicción a la evolución tendencial de la producción.
def prod22(finca, variedad, modo, campañas, pesos, df=df_group):
    prod = [df.loc[('PRODUCCION',c), (finca, variedad, modo)] for c in campañas]
    prod = [p for p in prod if p > 0]
    if len(prod) == 0:
        media_general = df.loc[:, (finca, variedad)].values.mean()
        media_general_total = np.nanmean(df.loc[:, (finca)].values)
        if media_general > 0:
            produccion22 = media_general
        elif media_general_total > 0:
            produccion22 = 3/4*min(media_general_total, 15000) # Si la finca no␣
 ↪producido esta
            # variedad pero sí otras, imputamos el 75% de su producción media␣
 ↪por variedad hasta
            # una producción media máxima de 15.000. La justificación de estos␣
 ↪valores se ha
            # realizado en la celda anterior
        else:
            produccion22 = 6311 # Si la finca no ha producido nunca ninguna␣
 ↪variedad imputamos la
            # esperanza calculada anteriormente
    else:
        pesos_prod = [p * w for p, w in zip(prod, pesos)]
        produccion22 = sum(pesos_prod) / sum(pesos[:len(prod)])
    produccion22 = round(produccion22, 2)
    return produccion22
```

[8]:
```
# Generamos la lista 'campañas', que recoge las campañas a tener en cuenta para␣
 ↪las ponderaciones de
# la función anterior. No se incluyen las campañas 14 y 15 por falta de datos␣
 ↪meteorológicos.
campañas = [16, 17, 18, 19, 20, 21]
```

[9]:
```
# Guardamos las predicciones en el dataframe df22 utilizando la función prod22,␣
 ↪la lista 'campañas'
# y el diccionario 'ponderaciones'.
```

```
for fila in range(8526,9601):
    df22.loc[fila, 'PRODUCCION'] = prod22(df.loc[fila, 'ID_FINCA'], df.
 ↪loc[fila, 'VARIEDAD'], df.loc[fila, 'MODO'], campañas, ponderaciones[df.
 ↪loc[fila, 'ID_ESTACION']])
```

[10]:
```
# Ajustamos el formato a las condiciones requeridas y guardamos el dataframe␣
 ↪resultante en un
# archivo .txt
df22.drop(['CAMPAÑA', 'ID_ZONA', 'ID_ESTACION', 'ALTITUD'], axis=1,␣
 ↪inplace=True)
df22['VARIEDAD']=df22['VARIEDAD'].astype(str)
df22['VARIEDAD']="0" + df22['VARIEDAD']
df22=df22.sort_values(['ID_FINCA', 'VARIEDAD', 'MODO', 'TIPO', 'COLOR',␣
 ↪'SUPERFICIE'])
df22.to_csv('output.txt', sep='|', index=False, header=False)
```

[11]:
```
df22.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1075 entries, 9240 to 8539
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID_FINCA    1075 non-null   int64
 1   VARIEDAD    1075 non-null   object
 2   MODO        1075 non-null   int64
 3   TIPO        1075 non-null   int64
 4   COLOR       1075 non-null   int64
 5   SUPERFICIE  1075 non-null   float64
 6   PRODUCCION  1075 non-null   float64
dtypes: float64(2), int64(4), object(1)
memory usage: 67.2+ KB
```

[12]:
```
# Comprobamos que no hay valores missing. Hemos asignado una predicción a cada␣
 ↪fila.
df22.isna().sum()
```

[12]:
```
ID_FINCA      0
VARIEDAD      0
MODO          0
TIPO          0
COLOR         0
SUPERFICIE    0
PRODUCCION    0
dtype: int64
```

Los modelos 2, 3 y 4 presentan otras alternativas de predicción. Utilizan algoritmos de mayor complejidad, lo cual puede dar lugar a una mayor exactitud a cambio de perder interpretabilidad.

2 Otros modelos

2.1 Modelo 2

En el siguiente modelo XGBOOST usamos exclusivamente los datos contenidos en la base de train sin tener en cuenta la variable superficie

[13]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score
```

[14]:
```
df = pd.read_pickle('df.pkl')
```

[15]:
```
#Conjunto de datos de entrenamiento
df1 = df.drop('SUPERFICIE', axis=1)
df1421=df1[df1['CAMPAÑA'] != 22]
#Conjunto de datos de test
df22 = df1[df1['CAMPAÑA'] == 22]
```

[16]:
```
#Train y evaluación
X = df1421.drop('PRODUCCION', axis=1).values
y = df1421['PRODUCCION'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[17]:
```
import xgboost as xgb
```

[18]:
```
# Defino los hiperparámetros
params = {'objective': 'reg:squarederror',
          'max_depth': 6,
          'learning_rate': 0.1,
          'subsample': 0.5,
          'colsample_bytree': 0.5,
          'n_estimators': 100}

# Entreno el modelo
model = xgb.XGBRegressor(**params)
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred_test = model.predict(X_test)
```

[19]:
```
# Vemos como ha sido el entrenamiento
y_pred_train=model.predict(X_train)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)
```

```
print(f'RMSE en train: {rmse_train:.5f}')
print(f'MAE en train: {mae_train:.5f}')
print("R2 score:", r2)
```

```
RMSE en train: 7385.70312
MAE en train: 4347.64939
R2 score: 0.6817699067121203
```

[20]:
```
# En test
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print('RMSE:', rmse)
mae=mean_absolute_error(y_test, y_pred_test)
print('MAE:', mae)
r2 = r2_score(y_test, y_pred_test)
print("R2 score:", r2)
```

```
RMSE: 8905.305603044813
MAE: 5035.646735583654
R2 score: 0.591792904102554
```

2.2 Modelo 3

En los siguientes modelos usamos exclusivamente los datos contenidos en la base de train, donde imputamos los 0's de la variable superficie como se indicó en el análisis exploratorio de TRAIN

[21]:
```
df = pd.read_pickle('df.pkl')
```

[22]:
```
#Creamos un diccionario para ir guardando los resultados
results_test_1 = {
    'XGboost': {'RMSE': None, 'MAE': None, 'R2': None},
    'GradientBoost': {'RMSE': None, 'MAE': None, 'R2': None},
    'Red Neuronal': {'RMSE': None, 'MAE': None, 'R2': None},
    'Bagging': {'RMSE': None, 'MAE': None, 'R2': None}
}
results_train_1 = {
    'XGboost': {'RMSE': None, 'MAE': None, 'R2': None},
    'GradientBoost': {'RMSE': None, 'MAE': None, 'R2': None},
    'Red Neuronal': {'RMSE': None, 'MAE': None, 'R2': None},
    'Bagging': {'RMSE': None, 'MAE': None, 'R2': None}
}
```

Entrenando con los datos de las CAMPAÑAS 14-21

[23]:
```
#Conjunto de datos de entrenamiento
df1421=df[df['CAMPAÑA'] != 22]
#Conjunto de datos de test
df22 = df[df['CAMPAÑA'] == 22]
```

[24]:
```
#Train y evaluación
X = df1421.drop('PRODUCCION', axis=1).values
y = df1421['PRODUCCION'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

GRADIENT BOOSTING

[25]:
```
# Creamos el modelo de Gradient Boosting
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.15,␣
 ↪max_depth=6, random_state=42)

# Ajustamos el modelo con los datos de entrenamiento
gbr.fit(X_train, y_train)

y_pred_test=gbr.predict(X_test)
```

[26]:
```
# Vemos como ha sido el entrenamiento
y_pred_train=gbr.predict(X_train)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)
print(f'RMSE en train: {rmse_train:.5f}')
print(f'MAE en train: {mae_train:.5f}')
print("R2 score:", r2)
```

```
RMSE en train: 3182.43477
MAE en train: 2082.04738
R2 score: 0.9409150904825445
```

[27]:
```
#Las guardo en el diccionario
results_train_1['GradientBoost']['RMSE'] = rmse_train
results_train_1['GradientBoost']['MAE'] = mae_train
results_train_1['GradientBoost']['R2'] = r2
```

[28]:
```
# En test
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print('RMSE:', rmse)
mae=mean_absolute_error(y_test, y_pred_test)
print('MAE:', mae)
r2 = r2_score(y_test, y_pred_test)
print("R2 score:", r2)
```

```
RMSE: 5621.3962151639125
MAE: 3154.409804035481
R2 score: 0.8373435504862835
```

[29]:
```
#Las guardo en el diccionario
results_test_1['GradientBoost']['RMSE'] = rmse
```

```
results_test_1['GradientBoost']['MAE'] = mae
results_test_1['GradientBoost']['R2'] = r2
```

XGBOOST

[30]:
```
# Defino los hiperparámetros
params = {'objective': 'reg:squarederror',
          'max_depth': 6,
          'learning_rate': 0.1,
          'subsample': 0.5,
          'colsample_bytree': 0.5,
          'n_estimators': 100}

# Entreno el modelo
model = xgb.XGBRegressor(**params)
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred_test = model.predict(X_test)
```

[31]:
```
# Vemos como ha sido el entrenamiento
y_pred_train=model.predict(X_train)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)
print(f'RMSE en train: {rmse_train:.5f}')
print(f'MAE en train: {mae_train:.5f}')
print("R2 score:", r2)
```

```
RMSE en train: 4664.18502
MAE en train: 2805.48439
R2 score: 0.87308612803866
```

[32]:
```
#Las guardo en el diccionario
results_train_1['XGboost']['RMSE'] = rmse_train
results_train_1['XGboost']['MAE'] = mae_train
results_train_1['XGboost']['R2'] = r2
```

[33]:
```
# En test
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print('RMSE:', rmse)
mae=mean_absolute_error(y_test, y_pred_test)
print('MAE:', mae)
r2 = r2_score(y_test, y_pred_test)
print("R2 score:", r2)
```

```
RMSE: 5809.922374595249
MAE: 3399.3802489852674
R2 score: 0.826250503732484
```

[34]:
```
#Las guardo en el diccionario
results_test_1['XGboost']['RMSE'] = rmse
results_test_1['XGboost']['MAE'] = mae
results_test_1['XGboost']['R2'] = r2
```

BAGGING

[35]:
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
```

[36]:
```
# Seleccionamos como modelo base un árbol de decisión
base_model = DecisionTreeRegressor()

# Creamos modelo de Bagging
bagging_model = BaggingRegressor(base_estimator=base_model, n_estimators=100,␣
 ↪random_state=30)

# Entrenamos modelo con datos de entrenamiento
bagging_model.fit(X_train, y_train)

# Hacemos predicciones en conjunto de train
y_pred_train = bagging_model.predict(X_train)
rmse = mean_squared_error(y_train, y_pred_train, squared=False)
mae = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"R2:: {r2:.3f}")
```

```
RMSE: 2285.091
MAE: 1117.859
R2:: 0.970
```

[37]:
```
#Las guardo en el diccionario
results_train_1['Bagging']['RMSE'] = rmse_train
results_train_1['Bagging']['MAE'] = mae_train
results_train_1['Bagging']['R2'] = r2
```

[38]:
```
# Hacemos predicciones en conjunto de test
y_pred_train = bagging_model.predict(X_test)
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
mae = mean_absolute_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"R2:: {r2:.3f}")
```

```
RMSE: 5809.922
MAE: 3399.380
R2:: 0.826
```

[39]:
```
#Las guardo en el diccionario
results_test_1['Bagging']['RMSE'] = rmse
results_test_1['Bagging']['MAE'] = mae
results_test_1['Bagging']['R2'] = r2
```

RED NEURONAL

[40]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.metrics import MeanAbsoluteError, RootMeanSquaredError

# Definir la arquitectura de la red neuronal
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

# Crear las instancias de las métricas MAE y R2
mae = MeanAbsoluteError()
rmse = RootMeanSquaredError()

# Compilar el modelo con las métricas
model.compile(optimizer='adam', loss='mean_squared_error', metrics=[mae, rmse])

# Entrenar el modelo
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,␣
 ↪y_test))

# Evaluar el modelo en el conjunto de entrenamiento
loss, mae, rmse = model.evaluate(X_train, y_train)
r2 = 1 - (rmse**2 / tf.math.reduce_variance(y_train))
print("Pérdida en el conjunto de prueba:", loss)
print("RMSE en el conjunto de prueba", rmse)
print("MAE en el conjunto de prueba:", mae)
print("R2 en el conjunto de prueba:", r2)
```

```
Epoch 1/10
214/214 [==============================] - 1s 3ms/step - loss: 185167584.0000 -
mean_absolute_error: 7948.3604 - root_mean_squared_error: 13607.6299 - val_loss:
210246000.0000 - val_mean_absolute_error: 9376.6621 -
val_root_mean_squared_error: 14499.8623
Epoch 2/10
214/214 [==============================] - 0s 2ms/step - loss: 176104288.0000 -
mean_absolute_error: 7913.8901 - root_mean_squared_error: 13270.4287 - val_loss:
```

```
207987584.0000 - val_mean_absolute_error: 7592.4629 -
val_root_mean_squared_error: 14421.7744
Epoch 3/10
214/214 [==============================] - 0s 2ms/step - loss: 172923168.0000 -
mean_absolute_error: 7949.9414 - root_mean_squared_error: 13150.0254 - val_loss:
193983952.0000 - val_mean_absolute_error: 8721.8955 -
val_root_mean_squared_error: 13927.8125
Epoch 4/10
214/214 [==============================] - 0s 2ms/step - loss: 171956160.0000 -
mean_absolute_error: 8036.9160 - root_mean_squared_error: 13113.2051 - val_loss:
195662416.0000 - val_mean_absolute_error: 7964.4932 -
val_root_mean_squared_error: 13987.9385
Epoch 5/10
214/214 [==============================] - 0s 2ms/step - loss: 170409232.0000 -
mean_absolute_error: 7953.6216 - root_mean_squared_error: 13054.0889 - val_loss:
193063376.0000 - val_mean_absolute_error: 8728.7480 -
val_root_mean_squared_error: 13894.7246
Epoch 6/10
214/214 [==============================] - 0s 2ms/step - loss: 170463984.0000 -
mean_absolute_error: 7975.7197 - root_mean_squared_error: 13056.1855 - val_loss:
193930000.0000 - val_mean_absolute_error: 8882.4131 -
val_root_mean_squared_error: 13925.8750
Epoch 7/10
214/214 [==============================] - 0s 2ms/step - loss: 170289072.0000 -
mean_absolute_error: 7966.9146 - root_mean_squared_error: 13049.4854 - val_loss:
192071920.0000 - val_mean_absolute_error: 8359.7188 -
val_root_mean_squared_error: 13859.0010
Epoch 8/10
214/214 [==============================] - 0s 2ms/step - loss: 170891616.0000 -
mean_absolute_error: 7995.4004 - root_mean_squared_error: 13072.5518 - val_loss:
194467360.0000 - val_mean_absolute_error: 9155.7471 -
val_root_mean_squared_error: 13945.1553
Epoch 9/10
214/214 [==============================] - 0s 2ms/step - loss: 170036144.0000 -
mean_absolute_error: 8003.9048 - root_mean_squared_error: 13039.7910 - val_loss:
195122096.0000 - val_mean_absolute_error: 7843.4009 -
val_root_mean_squared_error: 13968.6113
Epoch 10/10
214/214 [==============================] - 0s 2ms/step - loss: 168903120.0000 -
mean_absolute_error: 7964.1890 - root_mean_squared_error: 12996.2734 - val_loss:
194527616.0000 - val_mean_absolute_error: 7843.4727 -
val_root_mean_squared_error: 13947.3154
214/214 [==============================] - 0s 1ms/step - loss: 169717040.0000 -
mean_absolute_error: 7503.9004 - root_mean_squared_error: 13027.5488
Pérdida en el conjunto de prueba: 169717040.0
RMSE en el conjunto de prueba 13027.548828125
MAE en el conjunto de prueba: 7503.900390625
R2 en el conjunto de prueba: tf.Tensor(0.009891080799622864, shape=(),
```

```
dtype=float64)
```

[41]:
```
#Las guardo en el diccionario
results_train_1['Red Neuronal']['RMSE'] = rmse
results_train_1['Red Neuronal']['MAE'] = mae
results_train_1['Red Neuronal']['R2'] = r2
```

[42]:
```
# Evaluar el modelo en el conjunto de prueba
loss, mae, mse = model.evaluate(X_test, y_test)
r2 = 1 - (mse / tf.math.reduce_variance(y_test))
print("Pérdida en el conjunto de prueba:", loss)
print("MAE en el conjunto de prueba:", mae)
print("R2 en el conjunto de prueba:", r2)
```

```
54/54 [==============================] - 0s 1ms/step - loss: 194527616.0000 -
mean_absolute_error: 7843.4727 - root_mean_squared_error: 13947.3154
Pérdida en el conjunto de prueba: 194527616.0
MAE en el conjunto de prueba: 7843.47265625
R2 en el conjunto de prueba: tf.Tensor(0.999928208419033, shape=(),
dtype=float64)
```

[43]:
```
#Las guardo en el diccionario
results_test_1['Red Neuronal']['RMSE'] = rmse
results_test_1['Red Neuronal']['MAE'] = mae
results_test_1['Red Neuronal']['R2'] = r2
```

[44]:
```
#Nota:
#Aunque GradientBoost aporta errores de test menores, parece que los resultados␣
 ↪de XGboost
#caen menos en el sobreajuste al haber menores diferencias entre los errores de␣
 ↪entrenamiento y test
```

2.3 Modelo 4

En este modelo incluiremos variables meteorológicas correspondientes a los meses de enero-junio (meses más importantes en el ciclo de la vid)

[45]:
```
df = pd.read_pickle('df.pkl')
```

[46]:
```
import pickle

with open('rhum_mes_16.pkl', 'rb') as file:
    rhum_mes_16 = pickle.load(file)

with open('rhum_mes_17.pkl', 'rb') as file:
    rhum_mes_17 = pickle.load(file)

with open('rhum_mes_18.pkl', 'rb') as file:
```

```
    rhum_mes_18 = pickle.load(file)

with open('rhum_mes_19.pkl', 'rb') as file:
    rhum_mes_19 = pickle.load(file)

with open('rhum_mes_20.pkl', 'rb') as file:
    rhum_mes_20 = pickle.load(file)

with open('rhum_mes_21.pkl', 'rb') as file:
    rhum_mes_21 = pickle.load(file)

with open('rhum_mes_22.pkl', 'rb') as file:
    rhum_mes_22 = pickle.load(file)

with open('windspeed_mes_16.pkl', 'rb') as file:
    windspeed_mes_16 = pickle.load(file)

with open('windspeed_mes_17.pkl', 'rb') as file:
    windspeed_mes_17 = pickle.load(file)

with open('windspeed_mes_18.pkl', 'rb') as file:
    windspeed_mes_18 = pickle.load(file)

with open('windspeed_mes_19.pkl', 'rb') as file:
    windspeed_mes_19 = pickle.load(file)

with open('windspeed_mes_20.pkl', 'rb') as file:
    windspeed_mes_20 = pickle.load(file)

with open('windspeed_mes_21.pkl', 'rb') as file:
    windspeed_mes_21 = pickle.load(file)

with open('windspeed_mes_22.pkl', 'rb') as file:
    windspeed_mes_22 = pickle.load(file)

with open('t_mes_15.pkl', 'rb') as file:
    t_mes_15 = pickle.load(file)

with open('t_mes_16.pkl', 'rb') as file:
    t_mes_16 = pickle.load(file)

with open('t_mes_17.pkl', 'rb') as file:
    t_mes_17 = pickle.load(file)

with open('t_mes_18.pkl', 'rb') as file:
    t_mes_18 = pickle.load(file)
```

```
with open('t_mes_19.pkl', 'rb') as file:
    t_mes_19 = pickle.load(file)

with open('t_mes_20.pkl', 'rb') as file:
    t_mes_20 = pickle.load(file)

with open('t_mes_21.pkl', 'rb') as file:
    t_mes_21 = pickle.load(file)

with open('t_mes_22.pkl', 'rb') as file:
    t_mes_22 = pickle.load(file)

with open('precip_mes_16.pkl', 'rb') as file:
    precip_mes_16 = pickle.load(file)

with open('precip_mes_17.pkl', 'rb') as file:
    precip_mes_17 = pickle.load(file)

with open('precip_mes_18.pkl', 'rb') as file:
    precip_mes_18 = pickle.load(file)

with open('precip_mes_19.pkl', 'rb') as file:
    precip_mes_19 = pickle.load(file)

with open('precip_mes_20.pkl', 'rb') as file:
    precip_mes_20 = pickle.load(file)

with open('precip_mes_21.pkl', 'rb') as file:
    precip_mes_21 = pickle.load(file)

with open('precip_mes_22.pkl', 'rb') as file:
    precip_mes_22 = pickle.load(file)
```

[47]:
```
#Definimos una funcion para meter las variables meteorológicas de interés
def metervariables(df):
    meses=['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio']
    for m in range(len(meses)):
        precip_col = f"precip_{meses[m]}"
        df[precip_col] = np.nan

    # Rellenamos los valores de precip_mes en función de CAMPAÑA y ID_ESTACION
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                precip_mes = globals()[f"precip_mes_{c}"][e][m]
                mes_index = meses[m]
                precip_col = f"precip_{mes_index}"
```

```
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪precip_col] = precip_mes

    for m in range(len(meses)):
        windspeed_col = f"windspeed_{meses[m]}"
        df[windspeed_col] = np.nan

    # Rellenamos los valores de windspeed_mes en función de CAMPAÑA y␣
↪ID_ESTACION
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                windspeed_mes = globals()[f"windspeed_mes_{c}"][e][m]
                mes_index = meses[m]
                windspeed_col = f"windspeed_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪windspeed_col] = windspeed_mes

    # Rellenamos los valores de temp_mes en función de CAMPAÑA y ID_ESTACION
    for m in range(len(meses)):
        temp_col = f"t_mes_{meses[m]}"
        df[temp_col] = np.nan
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                temp_mes = globals()[f"t_mes_{c}"][e][m]
                mes_index = meses[m]
                temp_col = f"t_mes_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪temp_col] = temp_mes
    for m in range(len(meses)):
        rhum_col = f"rhum_mes_{meses[m]}"
        df[rhum_col] = np.nan
    for c in range(16, 23):
        for e in range(20):
            for m in range(6):
                rhum_mes = globals()[f"rhum_mes_{c}"][e][m]
                mes_index = meses[m]
                rhum_col = f"rhum_mes_{mes_index}"
                df.loc[(df['CAMPAÑA'] == c) & (df['ID_ESTACION'] == e),␣
↪rhum_col] = rhum_mes
    return df
```

[48]:
```
df2=metervariables(df)
```

[49]:
```
#Tenemos que eliminar los años de los que no disponemos datos meteorológicos
campanas_eliminar = [14, 15, 22]
df2 = df2[~df2['CAMPAÑA'].isin(campanas_eliminar)]
```

[50]:
```
#Creamos un diccionario para ir guardando los resultados
results_test_2 = {
    'XGboost': {'RMSE': None, 'MAE': None, 'R2': None},
    'GradientBoost': {'RMSE': None, 'MAE': None, 'R2': None},
    'Red Neuronal': {'RMSE': None, 'MAE': None, 'R2': None},
    'Bagging': {'RMSE': None, 'MAE': None, 'R2': None}
}
results_train_2 = {
    'XGboost': {'RMSE': None, 'MAE': None, 'R2': None},
    'GradientBoost': {'RMSE': None, 'MAE': None, 'R2': None},
    'Red Neuronal': {'RMSE': None, 'MAE': None, 'R2': None},
    'Bagging': {'RMSE': None, 'MAE': None, 'R2': None}
}
```

[51]:
```
#Train y evaluación
X = df2.drop('PRODUCCION', axis=1).values
y = df2['PRODUCCION'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[52]:
```
# Creamos el modelo de Gradient Boosting
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.15,␣
 ↪max_depth=6, random_state=42)

# Ajustamos el modelo con los datos de entrenamiento
gbr.fit(X_train, y_train)

y_pred_test=gbr.predict(X_test)
```

[53]:
```
# Vemos como ha sido el entrenamiento
y_pred_train=gbr.predict(X_train)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)
print(f'RMSE en train: {rmse_train:.5f}')
print(f'MAE en train: {mae_train:.5f}')
print("R2 score:", r2)
```

```
RMSE en train: 2684.87778
MAE en train: 1809.85876
R2 score: 0.9587569927348677
```

[54]:
```
#Las guardo en el diccionario
results_train_2['GradientBoost']['RMSE'] = rmse_train
results_train_2['GradientBoost']['MAE'] = mae_train
results_train_2['GradientBoost']['R2'] = r2
```

[55]:
```
# En test
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print('RMSE:', rmse)
mae=mean_absolute_error(y_test, y_pred_test)
print('MAE:', mae)
r2 = r2_score(y_test, y_pred_test)
print("R2 score:", r2)
```

```
RMSE: 5868.434744951996
MAE: 3280.535787839327
R2 score: 0.8126928635669151
```

[56]:
```
#Las guardo en el diccionario
results_test_2['GradientBoost']['RMSE'] = rmse
results_test_2['GradientBoost']['MAE'] = mae
results_test_2['GradientBoost']['R2'] = r2
```

XGBOOST

[57]:
```
# Defino los hiperparámetros
params = {'objective': 'reg:squarederror',
          'max_depth': 6,
          'learning_rate': 0.1,
          'subsample': 0.5,
          'colsample_bytree': 0.5,
          'n_estimators': 100}

# Entreno el modelo
model = xgb.XGBRegressor(**params)
model.fit(X_train, y_train)

# Realizamos predicciones en el conjunto de prueba
y_pred_test = model.predict(X_test)
```

[58]:
```
# Vemos como ha sido el entrenamiento
y_pred_train=model.predict(X_train)
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)
print(f'RMSE en train: {rmse_train:.5f}')
print(f'MAE en train: {mae_train:.5f}')
print("R2 score:", r2)
```

```
RMSE en train: 4303.45623
```

```
MAE en train: 2678.16757
R2 score: 0.8940414364389532
```

[59]:
```
#Las guardo en el diccionario
results_train_2['XGboost']['RMSE'] = rmse_train
results_train_2['XGboost']['MAE'] = mae_train
results_train_2['XGboost']['R2'] = r2
```

[60]:
```
# En test
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
print('RMSE:', rmse)
mae=mean_absolute_error(y_test, y_pred_test)
print('MAE:', mae)
r2 = r2_score(y_test, y_pred_test)
print("R2 score:", r2)
```

```
RMSE: 6451.814761651902
MAE: 3675.4190331559666
R2 score: 0.7736015002642348
```

[61]:
```
#Las guardo en el diccionario
results_test_2['XGboost']['RMSE'] = rmse
results_test_2['XGboost']['MAE'] = mae
results_test_2['XGboost']['R2'] = r2
```

BAGGING

[62]:
```
# Seleccionamos como modelo base un árbol de decisión
base_model = DecisionTreeRegressor()

# Creamos modelo de Bagging
bagging_model = BaggingRegressor(base_estimator=base_model, n_estimators=100,␣
 ↪random_state=30)

# Entrenamos modelo con datos de entrenamiento
bagging_model.fit(X_train, y_train)

# Hacemos predicciones en conjunto de train
y_pred_train = bagging_model.predict(X_train)
rmse = mean_squared_error(y_train, y_pred_train, squared=False)
mae = mean_absolute_error(y_train, y_pred_train)
r2 = r2_score(y_train, y_pred_train)

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"R2:: {r2:.3f}")
```

```
RMSE: 2549.294
MAE: 1245.888
R2:: 0.963
```

[63]:
```
#Las guardo en el diccionario
results_train_2['Bagging']['RMSE'] = rmse_train
results_train_2['Bagging']['MAE'] = mae_train
results_train_2['Bagging']['R2'] = r2
```

[64]:
```
# Hacemos predicciones en conjunto de test
y_pred_train = bagging_model.predict(X_test)
rmse = mean_squared_error(y_test, y_pred_test, squared=False)
mae = mean_absolute_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"R2:: {r2:.3f}")
```

```
RMSE: 6451.815
MAE: 3675.419
R2:: 0.774
```

[65]:
```
#Las guardo en el diccionario
results_test_2['Bagging']['RMSE'] = rmse
results_test_2['Bagging']['MAE'] = mae
results_test_2['Bagging']['R2'] = r2
```

RED NEURONAL

[66]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.metrics import RootMeanSquaredError

# Definir la arquitectura de la red neuronal
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

mae = MeanAbsoluteError()
rmse = RootMeanSquaredError()

# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error', metrics=[mae, rmse])

# Entrenar el modelo
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,␣
 ↪y_test))

# Evaluar el modelo en el conjunto de entrenamiento
```

```
loss, mae, rmse = model.evaluate(X_train, y_train)
r2 = 1 - (rmse**2 / tf.math.reduce_variance(y_train))
print("Pérdida en el conjunto de prueba:", loss)
print("RMSE en el conjunto de prueba", rmse)
print("MAE en el conjunto de prueba:", mae)
print("R2 en el conjunto de prueba:", r2)
```

```
Epoch 1/10
157/157 [==============================] - 1s 3ms/step - loss: 192170208.0000 -
mean_absolute_error: 8141.9038 - root_mean_squared_error: 13862.5469 - val_loss:
194459232.0000 - val_mean_absolute_error: 8095.1006 -
val_root_mean_squared_error: 13944.8643
Epoch 2/10
157/157 [==============================] - 0s 2ms/step - loss: 183338272.0000 -
mean_absolute_error: 8033.3809 - root_mean_squared_error: 13540.2461 - val_loss:
200735904.0000 - val_mean_absolute_error: 7378.9683 -
val_root_mean_squared_error: 14168.1299
Epoch 3/10
157/157 [==============================] - 0s 2ms/step - loss: 180125056.0000 -
mean_absolute_error: 8028.7554 - root_mean_squared_error: 13421.0674 - val_loss:
184792384.0000 - val_mean_absolute_error: 8845.5693 -
val_root_mean_squared_error: 13593.8359
Epoch 4/10
157/157 [==============================] - 0s 2ms/step - loss: 176279232.0000 -
mean_absolute_error: 8126.8916 - root_mean_squared_error: 13277.0186 - val_loss:
182387696.0000 - val_mean_absolute_error: 8319.2080 -
val_root_mean_squared_error: 13505.0986
Epoch 5/10
157/157 [==============================] - 0s 2ms/step - loss: 175158384.0000 -
mean_absolute_error: 8107.1665 - root_mean_squared_error: 13234.7412 - val_loss:
186928672.0000 - val_mean_absolute_error: 7799.2856 -
val_root_mean_squared_error: 13672.1865
Epoch 6/10
157/157 [==============================] - 0s 2ms/step - loss: 176064048.0000 -
mean_absolute_error: 8133.1533 - root_mean_squared_error: 13268.9131 - val_loss:
183889632.0000 - val_mean_absolute_error: 8085.2163 -
val_root_mean_squared_error: 13560.5908
Epoch 7/10
157/157 [==============================] - 0s 2ms/step - loss: 174056608.0000 -
mean_absolute_error: 8148.6431 - root_mean_squared_error: 13193.0518 - val_loss:
183814144.0000 - val_mean_absolute_error: 8026.3696 -
val_root_mean_squared_error: 13557.8076
Epoch 8/10
157/157 [==============================] - 0s 2ms/step - loss: 174995504.0000 -
mean_absolute_error: 8151.6709 - root_mean_squared_error: 13228.5869 - val_loss:
181810384.0000 - val_mean_absolute_error: 8541.2686 -
val_root_mean_squared_error: 13483.7080
Epoch 9/10
```

```
157/157 [==============================] - 0s 2ms/step - loss: 173892736.0000 -
mean_absolute_error: 8121.1138 - root_mean_squared_error: 13186.8398 - val_loss:
181533392.0000 - val_mean_absolute_error: 8363.7725 -
val_root_mean_squared_error: 13473.4326
Epoch 10/10
157/157 [==============================] - 0s 2ms/step - loss: 173504368.0000 -
mean_absolute_error: 8098.5308 - root_mean_squared_error: 13172.1055 - val_loss:
181398480.0000 - val_mean_absolute_error: 8500.2520 -
val_root_mean_squared_error: 13468.4258
157/157 [==============================] - 0s 1ms/step - loss: 172353008.0000 -
mean_absolute_error: 8273.7891 - root_mean_squared_error: 13128.3281
Pérdida en el conjunto de prueba: 172353008.0
RMSE en el conjunto de prueba 13128.328125
MAE en el conjunto de prueba: 8273.7890625
R2 en el conjunto de prueba: tf.Tensor(0.013901887220463927, shape=(),
dtype=float64)
```

[67]:
```python
#Las guardo en el diccionario
results_train_2['Red Neuronal']['RMSE'] = rmse
results_train_2['Red Neuronal']['MAE'] = mae
results_train_2['Red Neuronal']['R2'] = r2
```

[68]:
```python
# Evaluar el modelo en el conjunto de prueba
loss, mae, rmse = model.evaluate(X_test, y_test)
r2 = 1 - (rmse**2 / tf.math.reduce_variance(y_test))
print("Pérdida en el conjunto de prueba:", loss)
print("RMSE en el conjunto de prueba", rmse)
print("MAE en el conjunto de prueba:", mae)
print("R2 en el conjunto de prueba:", r2)
```

```
40/40 [==============================] - 0s 1ms/step - loss: 181398480.0000 -
mean_absolute_error: 8500.2520 - root_mean_squared_error: 13468.4258
Pérdida en el conjunto de prueba: 181398480.0
RMSE en el conjunto de prueba 13468.42578125
MAE en el conjunto de prueba: 8500.251953125
R2 en el conjunto de prueba: tf.Tensor(0.013394710017420763, shape=(),
dtype=float64)
```

[69]:
```python
#Las guardo en el diccionario
results_test_2['Red Neuronal']['RMSE'] = rmse
results_test_2['Red Neuronal']['MAE'] = mae
results_test_2['Red Neuronal']['R2'] = r2
```

[70]:
```python
print(results_test_1)
print(results_test_2)
print(results_train_2)
```

```
{'XGboost': {'RMSE': 5809.922374595249, 'MAE': 3399.3802489852674, 'R2':
0.826250503732484}, 'GradientBoost': {'RMSE': 5621.3962151639125, 'MAE':
```

3154.409804035481, 'R2': 0.8373435504862835}, 'Red Neuronal': {'RMSE':
13027.548828125, 'MAE': 7843.47265625, 'R2': <tf.Tensor: shape=(),
dtype=float64, numpy=0.999928208419033>}, 'Bagging': {'RMSE': 5809.922374595249,
'MAE': 3399.3802489852674, 'R2': 0.826250503732484}}
{'XGboost': {'RMSE': 6451.814761651902, 'MAE': 3675.4190331559666, 'R2':
0.7736015002642348}, 'GradientBoost': {'RMSE': 5868.434744951996, 'MAE':
3280.535787839327, 'R2': 0.8126928635669151}, 'Red Neuronal': {'RMSE':
13468.42578125, 'MAE': 8500.251953125, 'R2': <tf.Tensor: shape=(),
dtype=float64, numpy=0.013394710017420763>}, 'Bagging': {'RMSE':
6451.814761651902, 'MAE': 3675.4190331559666, 'R2': 0.7736015002642348}}
{'XGboost': {'RMSE': 4303.456228639298, 'MAE': 2678.167567581346, 'R2':
0.8940414364389532}, 'GradientBoost': {'RMSE': 2684.877783802826, 'MAE':
1809.8587634285097, 'R2': 0.9587569927348677}, 'Red Neuronal': {'RMSE':
13128.328125, 'MAE': 8273.7890625, 'R2': <tf.Tensor: shape=(), dtype=float64,
numpy=0.013901887220463927>}, 'Bagging': {'RMSE': 4303.456228639298, 'MAE':
2678.167567581346, 'R2': 0.9628172907160456}}

Vemos que, incorporar las variables meteorológicas de precipitaciones, humedad, temperatura y fuerza del viento de los años de los que disponemos datos (16-21) y eliminar los años 14 y 15, aporta peores resultados que trabajar exclusivamente con el dataset de TRAIN imputando SUPERFICIE Y ALTITUD. LLegados a este punto, decidimos intentar codificar las variables CAMPAÑA o ID_ESTACION o ambas usando las variables meteorológicas.La codificación empleando los datos meteorológicos nos permitirá trabajar con datos de menor dimensión. Así, vamos a construir una serie de modelos empleando este enfoque.