2_SolarGeneration_Modelacion

November 30, 2024

En este notebook se pretende hacer una predicción de la cantidad de Watts que se generaran en un dia en base a la cantidad de radiación solar que se recibe en un dia. Para esto se utilizaran varios modelos de regresión para ver cual es el que mejores resultados nos otorga.

1 Carga de Datos

[1]:
```
import pandas as pd
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error, median_absolute_error

import lightgbm as lgb
import xgboost as xgb

warnings.filterwarnings("ignore")
```

Cargamos los datos y los visualizamos

[2]:
```
df = pd.read_csv("Generation_data.csv")
df.head()
```

[2]:
```
   MODULE_TEMP  Amb_Temp  WIND_Speed  IRR (W/m2)  DC Current in Amps  \
0      18.7675  17.85190    47.60506    6.388252                0.60
1      18.6150  18.59573    64.26684   12.776500                0.66
2      18.9200  18.59573    85.68912   17.035340                4.74
3      18.9200  18.59573    83.30886   25.553010                8.18
4      19.0725  18.59573    57.12608   36.200090               26.66

   AC Ir in Amps  AC Iy in Amps  AC Ib in Amps  AC Power in Watts
0            8.6            8.6            8.7               3233
1            9.6            9.7           10.0               4504
```

```
2           11.9          12.0          12.4               6614
3           14.8          14.7          14.7               8971
4           18.6          18.4          18.5              12071
```

En el análisis previo, con la matriz de correlación, detectamos las variables que tienen una correlación alta con la variable objetivo, en este caso, la variable "AC Power in Watts".

Del analisis previo, sabemos que no hay valores nulos en el dataset, por lo que no es necesario hacer un tratamiento de valores nulos ni la imputacion de datos. Tambien sabemos que no hay variables categoricas, por lo que no es necesario hacer un tratamiento de variables categoricas. Y por ultimo, sabemos cuales son las variables mas relevantes.

[3]:
```
# Mantener solo las columnas relevantes
df = df[['MODULE_TEMP', 'Amb_Temp', 'IRR (W/m2)', 'AC Power in Watts']]
df.head()
```

[3]:
```
   MODULE_TEMP  Amb_Temp  IRR (W/m2)  AC Power in Watts
0      18.7675  17.85190    6.388252               3233
1      18.6150  18.59573   12.776500               4504
2      18.9200  18.59573   17.035340               6614
3      18.9200  18.59573   25.553010               8971
4      19.0725  18.59573   36.200090              12071
```

2 Separar las variables en X y Y

[4]:
```
# Separar las características (X) y la variable objetivo (y)
X = df[["MODULE_TEMP", "Amb_Temp", "IRR (W/m2)"]]
y = df["AC Power in Watts"]
```

3 Normalización de Datos

Utilizaremos la normalización de datos para que los modelos de regresión puedan trabajar de manera más eficiente. Probaremos con StandardScaler y MinMaxScaler para ver cual de los dos nos da mejores resultados.

[5]:
```
# Crear el scaler
scaler = StandardScaler()

# Ajustar y transformar los datos
X_scaled = scaler.fit_transform(X)

# Convertir el array escalado de nuevo a un DataFrame
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Mostrar los primeros registros del DataFrame escalado
X_scaled.head()
```

[5]:

	MODULE_TEMP	Amb_Temp	IRR (W/m2)
0	-1.528927	-1.303068	-1.346098
1	-1.541617	-1.113392	-1.325707
2	-1.516238	-1.113392	-1.312112
3	-1.516238	-1.113392	-1.284924
4	-1.503548	-1.113392	-1.250938

4 Modelacion y Entrenamiento

Para la modelacion y entrenamiento, se utilizaran los siguientes modelos de regresion: * Linear Regression * Polynomial Regression * Random Forest * XGBoost * LightGBM * Support Vector Machines (SVM)

5 Dividir el dataset en Train y Test

[6]:

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

5.1 Linear Regression

[7]:

```
# Crear el modelo de regresión lineal
model = LinearRegression()

# Entrenar el modelo
model.fit(X_train, y_train)

# Hacer predicciones
y_pred = model.predict(X_test)

# Evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
medae = median_absolute_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
print(f"Mean Absolute Error: {mae}")
print(f"Median Absolute Error: {medae}")
```

```
Mean Squared Error: 159747971.16967878
R^2 Score: 0.9808629271950605
Mean Absolute Error: 7473.24538237479
Median Absolute Error: 4958.310957427253
```

6 Polynomial Regression

[ ]:
```
# Crear el transformador de características polinómicas
poly = PolynomialFeatures(degree=2)

# Crear el modelo de regresión lineal
linear = LinearRegression()

# Crear el pipeline que primero transforma las características y luego ajusta
↪el modelo
model_poly = make_pipeline(poly, linear)

# Entrenar el modelo
model_poly.fit(X_train, y_train)

# Hacer predicciones
y_pred_poly = model_poly.predict(X_test)

# Evaluar el modelo
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)
mae_poly = mean_absolute_error(y_test, y_pred_poly)
medae_poly = median_absolute_error(y_test, y_pred_poly)

print(f"Mean Squared Error (Polynomial Regression): {mse_poly}")
print(f"R^2 Score (Polynomial Regression): {r2_poly}")
print(f"Mean Absolute Error (Polynomial Regression): {mae_poly}")
print(f"Median Absolute Error (Polynomial Regression): {medae_poly}")
```

```
Mean Squared Error (Polynomial Regression): 140500997.19574532
R^2 Score (Polynomial Regression): 0.9831686262253331
Mean Absolute Error (Polynomial Regression): 6821.709353959054
Median Absolute Error (Polynomial Regression): 4211.522203715169
```

7 Random Forest

[8]:
```
# Crear el modelo de Random Forest
rf_model = RandomForestRegressor(n_estimators=150, random_state=42)

# Entrenar el modelo
rf_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_rf = rf_model.predict(X_test)

# Evaluar el modelo
mse_rf = mean_squared_error(y_test, y_pred_rf)
```

```
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
medae_rf = median_absolute_error(y_test, y_pred_rf)

print(f"Mean Squared Error (Random Forest): {mse_rf}")
print(f"R^2 Score (Random Forest): {r2_rf}")
print(f"Mean Absolute Error (Random Forest): {mae_rf}")
print(f"Median Absolute Error (Random Forest): {medae_rf}")
```

```
Mean Squared Error (Random Forest): 163281125.82455277
R^2 Score (Random Forest): 0.9804396715044476
Mean Absolute Error (Random Forest): 7449.578253928051
Median Absolute Error (Random Forest): 4566.7236507936905
```

7.1 Busqueda de Hiperparametros para Random Forest

[9]:
```
# Definir los parámetros que queremos probar
param_grid = {
  'n_estimators': [100, 150, 200],
  'max_depth': [None, 10, 20, 30],
  'min_samples_split': [2, 5, 10],
  'min_samples_leaf': [1, 2, 4]
}

# Crear el modelo de Random Forest
rf_model = RandomForestRegressor(random_state=42)

# Crear el GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3,␣
 ↪n_jobs=-1, verbose=2)

# Entrenar el modelo
grid_search.fit(X_train, y_train)

# Obtener los mejores parámetros
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

# Evaluar el modelo con los mejores parámetros
best_rf_model = grid_search.best_estimator_
y_pred_best_rf = best_rf_model.predict(X_test)

# Evaluar el modelo
mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
r2_best_rf = r2_score(y_test, y_pred_best_rf)
mae_best_rf = mean_absolute_error(y_test, y_pred_best_rf)
medae_best_rf = median_absolute_error(y_test, y_pred_best_rf)
```

```python
print(f"Mean Squared Error (Best Random Forest): {mse_best_rf}")
print(f"R^2 Score (Best Random Forest): {r2_best_rf}")
print(f"Mean Absolute Error (Best Random Forest): {mae_best_rf}")
print(f"Median Absolute Error (Best Random Forest): {medae_best_rf}")
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split':
10, 'n_estimators': 200}
Mean Squared Error (Best Random Forest): 135551253.0213555
R^2 Score (Best Random Forest): 0.9837615828302749
Mean Absolute Error (Best Random Forest): 6606.933817275343
Median Absolute Error (Best Random Forest): 4125.318189967948
```

8 XGBoost

[10]:
```python
# Crear el modelo de XGBoost
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Entrenar el modelo
xgb_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_xgb = xgb_model.predict(X_test)

# Evaluar el modelo
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
medae_xgb = median_absolute_error(y_test, y_pred_xgb)

print(f"Mean Squared Error (XGBoost): {mse_xgb}")
print(f"R^2 Score (XGBoost): {r2_xgb}")
print(f"Mean Absolute Error (XGBoost): {mae_xgb}")
print(f"Median Absolute Error (XGBoost): {medae_xgb}")
```

```
Mean Squared Error (XGBoost): 137778087.35706922
R^2 Score (XGBoost): 0.9834948182106018
Mean Absolute Error (XGBoost): 6579.549506271723
Median Absolute Error (XGBoost): 4019.5625
```

9 LightGBM

[11]:
```python
# Crear el modelo de LightGBM
lgb_model = lgb.LGBMRegressor(random_state=42)

# Entrenar el modelo
```

```python
lgb_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_lgb = lgb_model.predict(X_test)

# Evaluar el modelo
mse_lgb = mean_squared_error(y_test, y_pred_lgb)
r2_lgb = r2_score(y_test, y_pred_lgb)
mae_lgb = mean_absolute_error(y_test, y_pred_lgb)
medae_lgb = median_absolute_error(y_test, y_pred_lgb)

print(f"Mean Squared Error (LightGBM): {mse_lgb}")
print(f"R^2 Score (LightGBM): {r2_lgb}")
print(f"Mean Absolute Error (LightGBM): {mae_lgb}")
print(f"Median Absolute Error (LightGBM): {medae_lgb}")
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000191 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 545
[LightGBM] [Info] Number of data points in the train set: 95092, number of used
features: 3
[LightGBM] [Info] Start training from score 127979.374890
Mean Squared Error (LightGBM): 133961829.8498358
R^2 Score (LightGBM): 0.983951988421836
Mean Absolute Error (LightGBM): 6521.984914138249
Median Absolute Error (LightGBM): 4032.6299075853167
```

10 Support Vector Machines (SVM)

[17]:
```python
# Crear el modelo de SVM
svm_model = SVR(kernel='rbf')

# Entrenar el modelo
svm_model.fit(X_train, y_train)

# Hacer predicciones
y_pred_svm = svm_model.predict(X_test)

# Evaluar el modelo
mse_svm = mean_squared_error(y_test, y_pred_svm)
r2_svm = r2_score(y_test, y_pred_svm)
mae_svm = mean_absolute_error(y_test, y_pred_svm)
medae_svm = median_absolute_error(y_test, y_pred_svm)
```

```python
print(f"Mean Squared Error (SVM): {mse_svm}")
print(f"R^2 Score (SVM): {r2_svm}")
print(f"Mean Absolute Error (SVM): {mae_svm}")
print(f"Median Absolute Error (SVM): {medae_svm}")
```

```
Mean Squared Error (SVM): 6096543796.479938
R^2 Score (SVM): 0.2696620705885764
Mean Absolute Error (SVM): 64262.76685718178
Median Absolute Error (SVM): 58646.782756845016
```