Ejercicio de Regresión en Series Temporales con Machine Learning

40GMAT

1 Introducción

Desarrollar un modelo de Machine Learning utilizando distintos métodos detallados a continuación para *predecir la temperatura basada en variables ambientales simuladas como la humedad y la precipitación.*

Este ejercicio tiene como objetivo implementar y comparar tres métodos de regresión en machine learning: Support Vector Regression (SVR), Kernel Ridge Regression (KRR) y Gaussian Processes (GP). Se deberá afinar los tres modelos para que minimice el error cuadrático medio (MSE) entre las predicciones y los valores reales.

2 Objetivo

Implementar y evaluar los modelos SVR, KRR y GP para predecir valores en una serie temporal compleja y comparar su rendimiento utilizando el error cuadrático medio (MSE).

3 Descripción de los Datos

- **Temperatura (T):**

$$T(t) = 20 + 10\sin\left(\frac{2\pi t}{50}\right) + \epsilon_T, \quad \epsilon_T \sim \mathcal{N}(0, 2)$$

- **Humedad (H):**

$$H(t) = 50 + 25\cos\left(\frac{2\pi t}{30}\right) + \epsilon_H, \quad \epsilon_H \sim \mathcal{N}(0, 5)$$

- **Precipitación (P):**

$$P(t) \sim \Gamma(2, 2)$$

4 Tareas

4.1 Generación de Datos

Generar 1000 puntos de datos utilizando la función descrita, con ruido introducido aleatoriamente en aproximadamente el 20% de los datos.

4.2 Preparación de Datos

Dividir los datos generados en un conjunto de entrenamiento (70%) y un conjunto de prueba (30%).

4.3 Implementación de Modelos

Implementar los siguientes modelos de regresión utilizando `scikit-learn`:

- Support Vector Regression (SVR)
- Kernel Ridge Regression (KRR)
- Gaussian Processes (GP)

Configurar hiperparámetros básicos para cada modelo.

4.4 Entrenamiento de Modelos

Entrenar cada modelo con el conjunto de datos de entrenamiento.

4.5 Evaluación de Modelos

Evaluar cada modelo en el conjunto de prueba y calcular el error cuadrático medio (MSE) para cada uno.

4.6 Visualización

Graficar tanto los datos originales como las predicciones de cada modelo para comparar visualmente sus rendimientos.

4.7 Análisis

Discutir el rendimiento de los modelos, indicando cuál fue el más efectivo y por qué.

5 Entregables

- Un informe escrito que discuta los resultados de los modelos, mencionando sus fortalezas y debilidades en este contexto específico.
- Código fuente en Python que realice todas las tareas enumeradas.
- Gráficos de salida que muestren las comparaciones de los modelos.

~\Desktop\Ruben\Machine_Learning_II\prueba_portafolio.py

```
1  ######################################
2  #Asignatura: Machine Learning II
3  #Autor: Rubén Garrido Hidalgo
4  #Curso: 4º
5  #Fecha: 29/04/2025
6  ######################################
7  ################################################################################ Portafolio_I
   ################################################################################
8
9  # ====================================================================
10 #4.1) Generar 1000 puntos de datos utilizando la funci ón descrita, con ruido introducido aleatoriamente en aproximadamente el 20% de los datos.
11 # ====================================================================
12 import numpy as np
13 import pandas as pd
14 from  sklearn  import  datasets
15 import matplotlib.pyplot as plt
16 from  sklearn.preprocessing  import  StandardScaler
17 from  sklearn.model_selection  import  train_test_split
18
19 #Generar 1000 puntos de datos utilizando la función descrita, con ruido introducido aleatoriamente en aproximadamente el 20% de los datos.
20 #Vamos a crear un dataset que poseerá tres columnas, las cuales serán [Temperatura, Humedad, precipitaciones].
21
22 #Realmente voy a  definir un dataset con la temperatura que es la variable dependiente y otro con las variables independientes que serán la humedad y las
   precipitaciones.
23
24 ### Generación de los datos:
25 #Temperatura: empelamos la fórmula T(t) = 20+10*sin (2πt) / 50 +ϵ_T, donde el ruído blanco se distribuye con una distribución normal de media cero y
   varianza 2, ϵ_T ~N(0,2).
26
27 #Para generar los 1000 datos vamos a emplear la funcion arange
28 numero_de_datos = 1000
29 datos_generados = np.arange(numero_de_datos)
30
31 ##########################################################################################################
32 #Datos con distribución Normal
33 #Definimos como se generan los datos de la serie temporal, excluyendo el ruído el cual será generado a parte:
34 datos_temperatura_sin_ruido = 20 + 10 * np.sin(2 * np.pi * datos_generados / 50)
35 datos_humedad_sin_ruido = 50 + 25 * np.cos(2 * np.pi * datos_generados / 30)
36
37 #Definimos el ruido de la temperatura y humedad. El ruído de cada una de estas series se distribuye con una media
38 #0 y una varianza determinada en el enunciado:
39 ruido_temperatura = np.random.normal(0,2,numero_de_datos)
40 ruido_humedad = np.random.normal(0,5,numero_de_datos)
41
42 #Una vez que tenemos definidos los términos generales de las series y el ruído de cada uan de ellas, vemos como
43 #para obtener la serie de datos que buscamos debemos realizar una combinación lineal entre los datos sin ruído y un 20%
44 #de los datos con el ruído previamente definido.
45
46 #Calculamos el 20% del ruído de cada serie:
47 ruido_20_por_ciento_temperatura = 0.2*(datos_temperatura_sin_ruido *ruido_temperatura)
48 ruido_20_por_ciento_humedad = 0.2*(datos_temperatura_sin_ruido *ruido_humedad)
49
50 #Por lo tanto la serie temporal definitivas para las variables temperatura y humedad serán:
51
52 #Temperatura
53 datos_temperatura_con_ruido = datos_temperatura_sin_ruido + ruido_20_por_ciento_temperatura
54 #Humedad
55 datos_humedad_con_ruido = datos_humedad_sin_ruido + ruido_20_por_ciento_humedad
56 #print(datos_humedad_con_ruido.ndim) #Veo la dimensión y es un ndarray de dimensión 1
57
58
59 #Visualizamos los datos (comento para que no salga siempre que cargamos el script)
60 #plt.plot(datos_generados, datos_temperatura_con_ruido, label= "Serie con ruído de la Temperatura", color= "black")
61 #plt.plot(datos_generados, datos_humedad_con_ruido, label= "Serie con ruído de la humedad", color= "red")
62 #plt.legend()
63 #plt.title("Representacion de la serie temporal")
64 #plt.grid()
65 #plt.show()
66 #plt.savefig("Visualización_series_estacionarias_1.png")
67
68
69 #Podemos observar que ambas series son estacionarias en media, pero en varianza no lo son del todo.
70
71 #A continuación para terminar de generar los datos de nuetro proyecto, vamos a generar los datos de las precipitaciones que se distribuyen con una
   distribución Gamma(2,2):
72
73 #Generamos los datos sin ruído
74 datos_precipitaciones_sin_ruido = np.random.gamma(shape = 2, scale = 2, size = numero_de_datos)
75
76 #Generamos el ruído de la serie
```

```
ruido_precipitaciones = np.random.gamma(shape = 2, scale = 2, size = numero_de_datos)

#Al no seguir una distribución normal hace que debamos normalizar los datos, para ello
#empelaremos la función StandardScaler la cual hemos importado al principio del script:

normalizador = StandardScaler()
ruido_precipitaciones_normalizado_2_dimensiones = ruido_precipitaciones.reshape(-1,1)
ruido_precipitaciones_normalizado = normalizador.fit_transform(ruido_precipitaciones_normalizado_2_dimensiones)

#Una vez que los tenemos normalizados, procedemos como antes:
ruido_20_por_ciento_precipitaciones = 0.2*(datos_precipitaciones_sin_ruido *ruido_precipitaciones_normalizado)

#Por lo tanto la serie temporal de las precipitaciones con ruído será:
datos_precipitaciones_con_ruido = datos_precipitaciones_sin_ruido + ruido_20_por_ciento_precipitaciones

#plt.plot(datos_generados, datos_precipitaciones_con_ruido, label= "Serie con ruído de las Precipitaciones", color= "black")
#plt.title("Representacion de la serie temporal de las Precipitaciones")
#plt.show()

#Por lo tanto una vez que tenemos creadas las tres series temporales con ruído vamos construir los datos para nuestro modelo de Aprendizaje Automático:

#Voy a crear un DataFrame para manipular los datos con la librería Pandas entre otras:
#Para ello debemos ajustar los datos para que no tengamos ningun error cuando creemos el DataFrame.

#Empleo la función flatten() para convertir estos arrays en arrays unidimensionales:
datos_generados_ajustados = datos_generados.flatten()
datos_temperatura_con_ruido_ajustados = datos_temperatura_con_ruido.flatten()
datos_humedad_con_ruido_ajustados = datos_humedad_con_ruido.flatten()
datos_precipitaciones_con_ruido_ajustados = datos_precipitaciones_con_ruido.flatten()

#La longitud de datos_precipitaciones_con_ruido_ajustados es de 1000000, por lo tanto vamos a ajustar la longitud para que sea
#igual a la de los otros datos con ruido ajustados.
datos_precipitaciones_con_ruido_ajustados_2 = np.resize(datos_precipitaciones_con_ruido, 1000)
#print(datos_precipitaciones_con_ruido_ajustados_2.shape)


datos_globales = pd.DataFrame({
        "Tiempo": datos_generados_ajustados,
        'Temperatura': datos_temperatura_con_ruido_ajustados,
        'Humedad': datos_humedad_con_ruido_ajustados,
        'Precipitaciones': datos_precipitaciones_con_ruido_ajustados_2
    })


#print(datos_globales.columns)

#Estructuramos los datos en la variable dependiente "y" y las variables independientes en X, para nuestro modelo:
X = datos_globales[["Temperatura", "Humedad"]]
y = datos_globales["Precipitaciones"]
#print(X)
#print(y)

# ======================================================================
# 4.2) Dividir los datos generados en un conjunto de entrenamiento (70%) y un conjunto de prueba (30%).
X_train, X_test, y_train, y_test  =  train_test_split(X,  y, test_size=0.3, random_state=42)
# ======================================================================

# ======================================================================
# 4.3) Implementación de los modelos
# ======================================================================
#Antes de aplicar los modelos y ajustar sus parámetros vamos a estandarizar los datos:
from  sklearn.preprocessing  import  StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ======================================================================
#Support Vector Regression (SVR)
# ======================================================================
#Importamos las librerías necesarias:
from  sklearn.svm  import  SVR
from  sklearn.model_selection  import  train_test_split, GridSearchCV
from  sklearn.preprocessing  import  StandardScaler
from  sklearn.metrics  import  mean_squared_error

#Definimos el modelo con los datos de entrenamiento:
#Sabemos que nuestro modelo de SVR posee dos parámetros, C y épsilon los cuales influyen directamente sobre
#como el modelo se ajusta a los datos, por lo tanto debemos modificarlos para minimizar el error cuadrático medio.


# Importamos la función GridSearchCV (Búsqueda de cuadrícula) de la librería sklearn.model_selection y procedemos a
```

```
#la búsqueda de los valores más óptimos de C y épsilon:

param_grid = {
"C": [0.1, 1, 10, 100],
"gamma": [1, 0.1, 0.01, 0.001],
"kernel": ["rbf", "poly", "sigmoid"]
}

grid = GridSearchCV(SVR(), param_grid, refit=True, cv=5)

#Al dar el valor cv = 5 implica que estamos realizando una cross- validation en la cual estamos dividiendo el conjunto de entrenamiento en cinco subconjuntos.

#Entrenamos este modelo y predecimos los mejores parámetros para el mismo:
grid.fit(X_train_scaled, y_train)
print(f"Mejores parámetros: {grid.best_params_}")

#Mejores parámetros: {'C': 0.1, 'gamma': 0.001, 'kernel': 'poly'}

#Nuestro modelo sería
#model_SVR_1 = SVR(kernel="poly",C = 0.1, gamma= 0.001, epsilon= 0.1)
#pero este modelo se subajusta, pues vemos como en la represnetación no captura las relaciones no lineales y que las predicciones y los
#datos originales forman una recta. Es decir, está subajustado

#Es por ello que con el objetivo de evitar el subajuste y vcapturar las relaciones no lienales del modelo vamos a implementar un kernel radial (rbf),
#y vamos a modificar los parámetros del modelo C y gamma.


#Primero probé este modelo:
model_SVR = SVR(kernel="rbf",C = 10, gamma="scale", epsilon= 0.1)



# ======================================================================
#Kernel Ridge Regression (KRR)
# ======================================================================

#Importamos las nuevas librerías necesarias:
from sklearn.kernel_ridge import KernelRidge

#Creamos el modelo de Kernel Ridge Regression
model_krr = KernelRidge() #optimizamos los parámetros del modelo como son el kernel empleado, gamma y alpha mediante la búsqueda por cuadrícula:
#Aquí adjunto un modelo que me sirve para relizar unas predicciones que hacen que la represnetacón sea más visual, la cual emplea un kernel rbf, pero que se sobreajusta al modelo.
model_krr_2 = KernelRidge(alpha= 0.3, kernel='rbf', gamma=5)
#Vamos a emplear la función param_dig para optimizar los parámetros del modelo KRR
param_grid = { "alpha":[0.1, 1, 10, 100], # Mide la regularización de los datos
              "kernel":['linear', 'polynomial','rbf', 'sigmoid'],
              "gamma":np.logspace(-4, 1,6), # Coeficientes del Kernel
              "degree":[2, 3, 4] #Grado de los polinomios en caso de ser un kernel polinomial
                }
grid_search = GridSearchCV(estimator=model_krr,param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)


# Entrenamos el grid_search para la búsqueda de los mejores parámetros alpha y gamma, así como el mejor kernel:
grid_search.fit(X_train_scaled,y_train)

# Extraemos los resultados:
scores = grid_search.cv_results_['mean_test_score']
gammas = grid_search.cv_results_['param_gamma']

# Cambiamos de signo los resultados:
scores = -scores

# Imprimimos los mejores parámetros y la mejor puntuación:
print("Mejores parámetros:", grid_search.best_params_) # --> #Mejores parámetros: {'alpha': 1, 'degree': 2, 'gamma': 0.0001, 'kernel': 'sigmoid'}
print("Mejor score:",-grid_search.best_score_) #--> #Mejor score: 9.199398566520903


# Extraemos los mejores parámetros
best_params= grid_search.best_params_

# Redefinimos nuetsro modelo KRR con los parámetros obtenidos anteriormente
best_krr_1= KernelRidge(**best_params)
#Este ajuste es malo y tras ajustar los parámetros vamos a ver que el mejor modelo es:
best_krr = KernelRidge(kernel='rbf', alpha=0.2, gamma=0.01)

# ======================================================================
#Procesos Gaussianos de Regresión (GP)
# ======================================================================

#Importamos las librerías necesarias
```

```python
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C, ExpSineSquared, WhiteKernel
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C, ExpSineSquared, WhiteKernel

#Para poder capturar la linealidad, la no lialidad, asi como al periodicidad o estacionalidad de la serie temporal vamos a emplear un kernel
#compuesto:
# Definimos un kernel compuesto con las siguientes funciones:
# DotProduct para capturar la tendencia lineal
# ExpSineSquared para la periodicidad
# RBF para variaciones suaves (relaciones no lineales)

#Este es le mejor kernel que he encontrado:
kernel_2 = C(1.0, (1e-2, 1e2)) * RBF(0.5, (1e-2, 1e2)) + ExpSineSquared(length_scale=1.0,
                                                                        periodicity=1.0,
                                                                        length_scale_bounds=(0.1, 10.0),
                                                                        periodicity_bounds=(0.5, 1.5))

#Definimos el modelo:
model_gpr = GaussianProcessRegressor(kernel=kernel_2, n_restarts_optimizer=10, alpha=1e-2, normalize_y=True)


# ======================================================================
#4.4) Entrenamiendo de los modelos
# ======================================================================

#Modelo SVR
#Entrenamos el modelo de Máquinas de Sopote Vectorial con los datos de entrenamiento
model_SVR.fit(X_train_scaled,y_train)

#Modelo KRR
# Entrenar el modelo  de Kernel Ridge Regression con los datos de entrenamiento
best_krr.fit(X_train_scaled, y_train)

#Entrenamos el Proceso Gaussiano de Regresión con los datos de entrenamiento
model_gpr.fit(X_train_scaled, y_train)

# ======================================================================================================================================================
#4.5)Evaluar cada modelo en el conjunto de prueba, realizamos las predicciones  y calculamos el error cuadratico medio (MSE) para cada uno.
# ======================================================================================================================================================
#Una vez que hemos entrenado los modelos vamos a relizar la predicción de los mismos sobre el conjunto de prueba ya escalado previamente (X_test_scaled):
#Predicciones de cada modelo:

prediccion_y_SVR = model_SVR.predict(X_test_scaled)
prediccion_y_KRR = best_krr.predict(X_test_scaled)
prediccion_y_PGR, sigma = model_gpr.predict(X_test_scaled, return_std=True)

#Para medir la precisión de la predicción con respecto a los valores originales de los datos calculamos como métrica el Error Cuadrático Medio (MSE) de cada
modelo:
mse_SVR = mean_squared_error(y_test,  prediccion_y_SVR)
print(f"Error Cuadrático Medio de la máquina de SVR:  {mse_SVR:.2f}")
mse_KRR = mean_squared_error(y_test, prediccion_y_KRR)
print(f"Error Cuadrático Medio del Kernel Ridge Regresssion,:", mse_KRR)
mse_PGR = mean_squared_error(y_test, prediccion_y_PGR)
print(f"Error Cuadrático Medio del Proceso Gaussiano,:", mse_PGR)

# ============================================================================================================================================================
#4.6)Graficar tanto los datos originales como las predicciones de cada modelo para comparar visualmente sus rendimientos.
# ============================================================================================================================================================

#Representación de las series temporales con ruído al 20% generadas en el aparatdo 4.1:
#Creamos la cuadrícula para la represnetación
plt.figure(figsize=(15, 6))

# Graficamos las series temporales correspondientes a la temperatura, humedad y precipitaciones
plt.plot(datos_generados, datos_temperatura_con_ruido, label="Temperatura", color="red", linewidth=1)

plt.plot(datos_generados, datos_humedad_con_ruido, label="Humedad", color="blue", linewidth=1)

plt.plot(datos_generados, datos_precipitaciones_con_ruido_ajustados_2, label="Precipitaciones", color="green", linewidth=1)

#Detalles de la gráfica, introducimos el título, el nombre de los ejes y la leyenda.
plt.title("Series Temporales con Ruido: Temperatura, Humedad y Precipitaciones")
plt.xlabel("Tiempo")
plt.ylabel("Valor de la variable")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ========================================
# 4.6.1. Visualización del KRR
# ========================================
```

```
320
321 import numpy as np
322 import matplotlib.pyplot as plt
323
324 #Reoresentación en 2D empleando las tres variables:
325
326 plt.figure(figsize=(14, 7))
327
328 # Representación de los valores reales con color según humedad
329 sc = plt.scatter(X_test["Temperatura"],y_test,c=X_test["Humedad"],cmap="plasma",label="Datos reales",alpha=0.5)
330
331 # Representación de las predicciones de KRR
332 plt.scatter(X_test["Temperatura"],prediccion_y_KRR,c=X_test["Humedad"],cmap="plasma",edgecolor="black",label="Predicción KRR",alpha=0.7)
333
334 plt.title("Predicciones con Kernel Ridge Regression (KRR) - Color según Humedad")
335 plt.xlabel("Temperatura")
336 plt.ylabel("Precipitaciones")
337 plt.grid(True)
338 plt.legend()
339
340 # Barra de color que indica los niveles de humedad
341 cbar = plt.colorbar(sc)
342 cbar.set_label("Humedad")
343
344 plt.tight_layout()
345 plt.savefig("predicciones_krr_con_humedad.png")
346 plt.show()
347
348 ##########################################################################################################################################################################
349 #Representación 3D de KRR
350 from mpl_toolkits.mplot3d import Axes3D  # Importamos para graficar en 3D
351
352 # Visualización 3D de la predicción del modelo
353 fig = plt.figure(figsize=(10, 8))
354 ax = fig.add_subplot(111, projection='3d')
355
356 # Visualizar los datos de entrenamiento
357 ax.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], y_train, color='darkorange', label='Training data', alpha=0.6)
358
359 # Visualizar los datos de prueba
360 ax.scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], y_test, color='gray', label='Test data', alpha=0.6)
361
362 # Visualizar el modelo predicho
363 # Creamos una malla para evaluar las predicciones
364 x1_grid, x2_grid = np.meshgrid(np.linspace(X_train_scaled[:, 0].min(), X_train_scaled[:, 0].max(), 100),
365                                np.linspace(X_train_scaled[:, 1].min(), X_train_scaled[:, 1].max(), 100))
366
367 # Predecir las precipitaciones para la malla de características
368 grid_data = np.array([x1_grid.ravel(), x2_grid.ravel()]).T
369 predictions = best_krr.predict(grid_data)
370
371 # Reformateamos las predicciones para que coincidan con la forma de la malla
372 predictions = predictions.reshape(x1_grid.shape)
373
374 # Graficamos el modelo en 3D
375 ax.plot_surface(x1_grid, x2_grid, predictions, color='navy', alpha=0.3, label='Best model')
376
377 # Etiquetas y título
378 ax.set_xlabel('Temperatura (normalizada)')
379 ax.set_ylabel('Humedad (normalizada)')
380 ax.set_zlabel('Precipitaciones')
381 ax.set_title('Kernel Ridge Regression (KRR) sobre los datos')
382
383 # Añadir la leyenda
384 ax.legend()
385
386 # Mostrar la gráfica
387 plt.tight_layout()
388 plt.show()
389 ##########################################################################################################################################################################
390
391 # ========================================
392 # 4.6.2. Visualización del proceso Gaussiano
393 # ========================================
394 plt.figure(figsize=(14, 7))
395
396 # Puntos reales, codificando la humedad como color
397 sc_1 = plt.scatter(X_test["Temperatura"],y_test,c=X_test["Humedad"],cmap='viridis',label='Datos reales',alpha=0.5)
398
399 # Predicciones
400 plt.scatter( X_test["Temperatura"],prediccion_y_PGR, c=X_test["Humedad"], cmap='viridis', edgecolor='k', label='Predicción GPR', alpha=0.5)
401
```

```python
# Intervalos de confianza
plt.errorbar(X_test["Temperatura"], prediccion_y_PGR, yerr=1.96 * sigma, fmt='o', color='gray', alpha=0.2, label='Intervalo 95%')

plt.title('Regresión con Proceso Gaussiano')
plt.xlabel('Temperatura')
plt.ylabel('Precipitaciones')
plt.grid(True)
plt.legend()

# Barra de colores para matizar el nivel de humedad
cbar = plt.colorbar(sc_1)
cbar.set_label('Humedad')

plt.tight_layout()
plt.savefig("proceso_gaussiano_resultado_con_humedad.png")
plt.show()

# ========================================
# 4.6.3. Visualización del SVR
# ========================================

#Represnetación en 2D:
plt.figure(figsize=(14, 7))

# Representación de los valores reales (color según humedad)
sc_2 = plt.scatter( X_test["Temperatura"], y_test, c=X_test["Humedad"], cmap="coolwarm", label="Datos reales",alpha=0.5)

# Representación de las predicciones SVR
plt.scatter(X_test["Temperatura"], prediccion_y_SVR, c=X_test["Humedad"], cmap="coolwarm", edgecolor="black", label="Predicción SVR", alpha=0.7)

plt.title("Predicciones SVR con codificación de color por Humedad")
plt.xlabel("Temperatura")
plt.ylabel("Precipitaciones")
plt.grid(True)
plt.legend()

# Barra de colores para matizar el nivel de humedad
cbar = plt.colorbar(sc_2)
cbar.set_label("Humedad")

plt.tight_layout()
plt.savefig("predicciones_svr_con_humedad.png")
plt.show()


# Visualización 3D de la predicción del modelo SVR
from mpl_toolkits.mplot3d import Axes3D  # Importamos para graficar en 3D


fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Visualizamos los datos de entrenamiento y prueba
ax.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], y_train, color='darkorange', label='Training data', alpha=0.6)
ax.scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], y_test, color='gray', label='Test data', alpha=0.6)

# Visualizamos el modelo predicho
# Creamos una malla para evaluar las predicciones
x1_grid, x2_grid = np.meshgrid(np.linspace(X_train_scaled[:, 0].min(), X_train_scaled[:, 0].max(), 100),
                               np.linspace(X_train_scaled[:, 1].min(), X_train_scaled[:, 1].max(), 100))

# Realizamos las precipitaciones para la malla de características
grid_data = np.array([x1_grid.ravel(), x2_grid.ravel()]).T
predictions = model_SVR.predict(grid_data)

#Ajustmos las predicciones para que coincidan con la forma de la malla
predictions = predictions.reshape(x1_grid.shape)

# Graficamos el modelo en 3D
ax.plot_surface(x1_grid, x2_grid, predictions, color='navy', alpha=0.3, label='RBF model')

# Etiquetamos todo, añadimos el título y la leyenda:
ax.set_xlabel('Temperatura (normalizada)')
ax.set_ylabel('Humedad (normalizada)')
ax.set_zlabel('Precipitaciones')
ax.set_title('Support Vector Regression (SVR) sobre Datos Meteorológicos')

ax.legend()

# Mostramos la gráfica
plt.tight_layout()
plt.show()
```

```
484
485
486 ##########################################################################################################################################################################
487 #Los apartados 4.7 y 5 se encuentran en el informe redactado.
```

Entrega I: Machine Learning II

Informe sobre la implementación de modelos de Aprendizaje Automático sobre datos generados por series temporales para predecir la temperatura basada en variables ambientales simuladas como la humedad y las precipitaciones.

Descripción del proyecto:

En este proyecto vamos a definir los datos siguiendo la descripción de los mismos dados en el enunciado, como series temporales. Posteriormente, tras un preprocesamiento de los datos implementaremos tres modelos de regresión de aprendizaje automático:

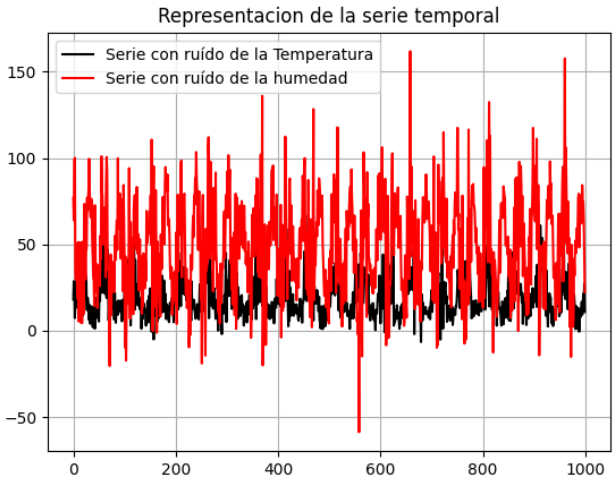
- Support Vector Regression (SVR)
- Kernel RidgeRegression (KRR)
- Gaussian Processes (GP)

Vamos a implementar distintos modelos de aprendizaje automático para realizar predicciones sobre nuestros datos, después de un proceso previo de entrenamiento. Una vez obtenidas las predicciones, evaluaremos su rendimiento utilizando el error cuadrático medio (MSE) como métrica principal, con el objetivo de minimizar error entre los datos de test y las predicciones.

Posteriormente, generaremos representaciones gráficas tanto de los modelos como de los datos originales, lo que nos permitirá comparar visualmente los resultados de cada modelo. Finalmente, analizaremos las predicciones realizadas por todos los modelos para determinar cuál de ellos ofrece el mejor desempeño sobre nuestros datos.

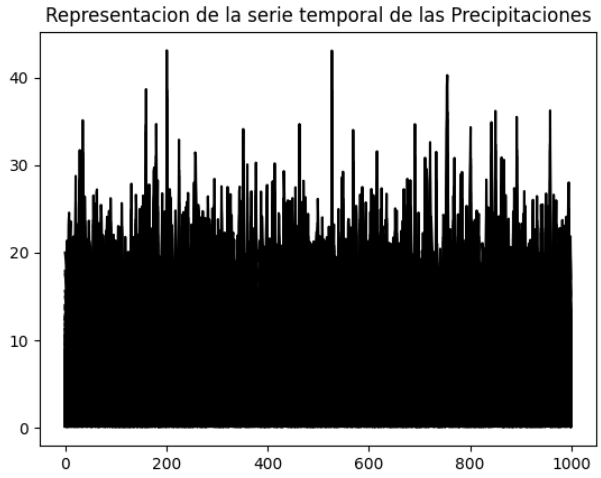
4.1) *Generar 1000 puntos de datos utilizando la función descrita, con ruido introducido aleatoriamente en aproximadamente el 20% de los datos.*

Empleando las definiciones del enunciado y diferentes funciones de las librerías que estamos usando hemos podido definir las series temporales de la temperatura y humedad que siguen una distribución normal:



Podemos observar que ambas series son estacionales en media, pero no en varianza excluyendo algunos outliers o valores atípicos que podemos observar, pero de forma esporádica.

A continuación, generamos los datos de la serie temporal sintética que nos falta, la serie temporal de las precipitaciones. Los datos de las precipitaciones a diferencia de los datos de la humedad y la temperatura se distribuyen con una distribución Gamma(2,2).



Una vez creadas las series temporales vamos a ajustar nuestros datos para crear un DataFrame y así poder aplicar nuestros modelos de Aprendizaje Automático. Se puede ver todos los ajustes en el documento adjuntado con el código.

Finalmente obtenemos dos arrays con los datos. Uno de ellos llamado “X”, el cual posee los datos de las variables independientes “Temperatura” y “Humedad”. Y otro nombrado como “y” que contiene la variable dependiente “Precipitaciones”:

```
[1000 rows x 2 columns]
0      1.058558
1      2.934869
2      8.002175
3      1.569030
4      5.289036
         ...
995    3.355047
996    1.908503
997    8.137133
998    5.140992
999    2.635362
Name: Precipitaciones, Length: 1000, dtype: float64
```

	Temperatura	Humedad
0	15.809081	84.572414
1	9.956078	74.045464
2	21.880808	77.390587
3	37.329930	85.836221
4	2.486691	62.140238
..	...	...
995	2.812850	76.576641
996	16.250823	49.195602
997	14.335542	56.158781
998	23.040769	65.869862
999	18.234824	46.137904

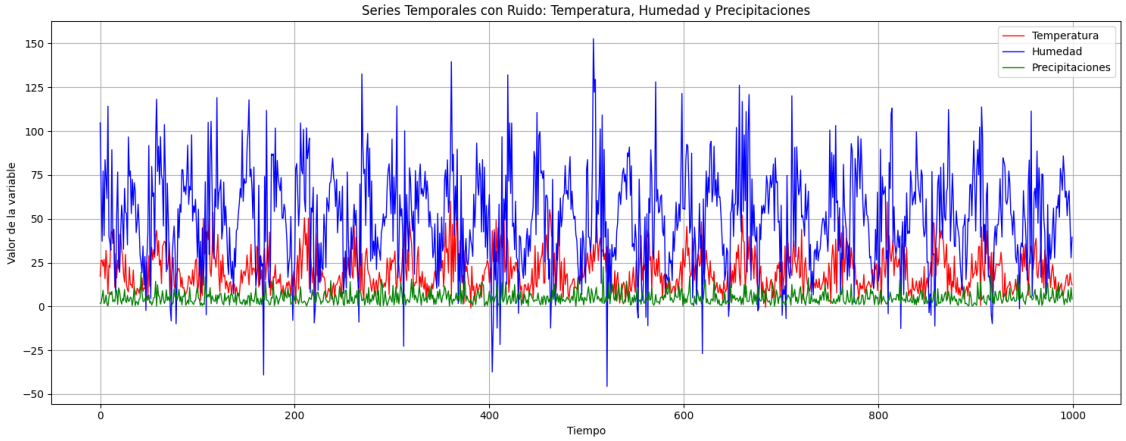
4.2) *Dividir los datos generados en un conjunto de entrenamiento (70%) y un conjunto de prueba (30%).*

Una vez hemos definido todos los datos que teníamos que generar dividimos estos datos en dos conjuntos, uno de entrenamiento que corresponderá al 70% de los datos y otro conjunto de prueba que corresponderá al 30% de los restantes.

```
X_train, X_test, y_train, y_test  =  train_test_split(X,  y, test_size=0.3, random_state=42)
```

Una vez divididos los datos generados en dos conjuntos para entrenamiento y prueba voy a implementar y analizar cada uno de los tres modelos especificados en el enunciado mencionando sus fortalezas y debilidades en el análisis de las series temporales ya construidas.

Representación de los datos originales:



En esta gráfica vemos representado con tres colores diferentes las tres series temporales con ruido que hemos construido. La serie de la humedad (azul), la temperatura (rojo) y las precipitaciones (verde).

En dicha representación podemos observar que la serie temporal con mayor varianza es la que corresponde a la humedad. Todas las series corresponden a series temporales estacionarias en media, pero no en varianza, como mencionamos anteriormente.

Comenzamos a continuación con la representación de cada uno de los modelos de aprendizaje automático:

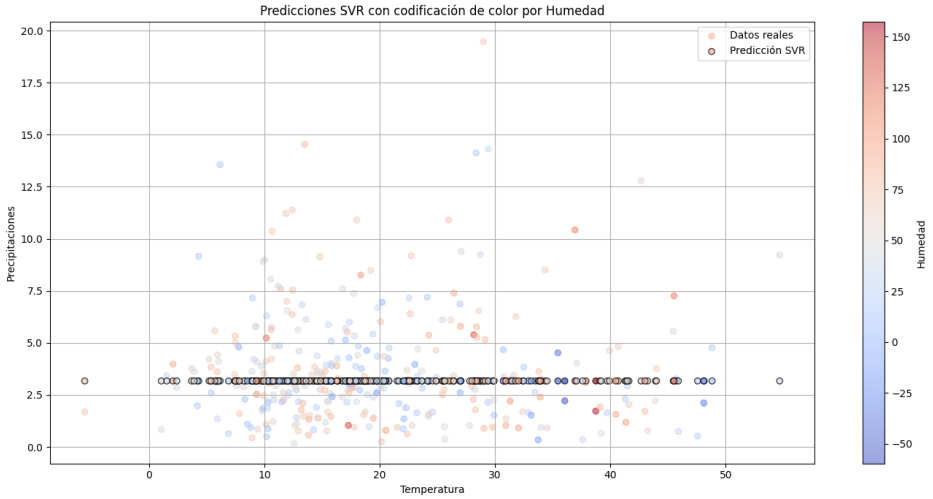
Como hemos definido en el temario el error cuadrático medio (MSE) se define como la diferencia entre las predicciones de un modelo y sus datos originales. En nuestros modelos de aprendizaje automático buscamos minimizar este error cuadrático medio, para poder encontrar un modelo que se ajuste lo mejor posible a los datos sin producir un overfitting y además realice las mejores predicciones posibles.

En este proyecto hemos implementado tres modelos:

- ## Support Vector Regression (SVR):

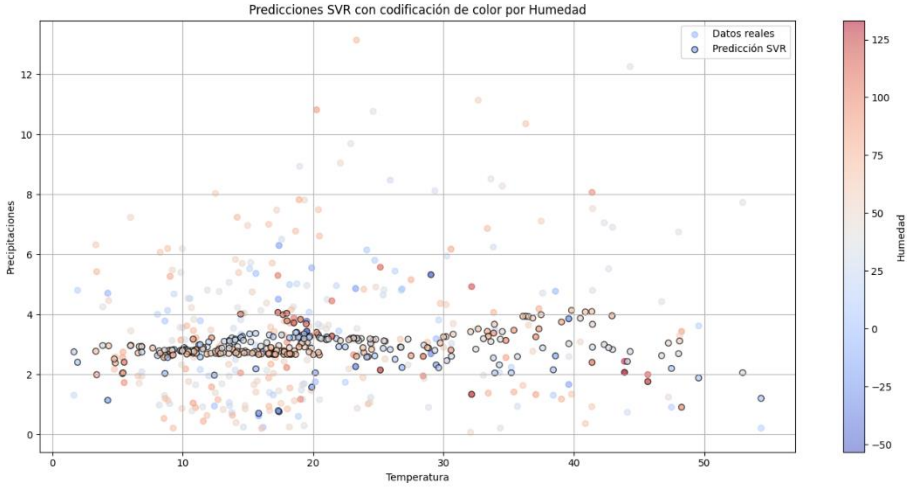
La regresión con Máquinas de Soporte es un método de regresión que emplea un kernel y posee una dependencia de ciertos parámetros para poder ajustar el modelo.

Inicialmente, se optó por un kernel polinómico con coeficientes C = 0,001 y épsilon = 0,1 tras observar su buen desempeño durante la fase de validación mediante búsqueda en malla (GridSearchCV). Sin embargo, al aplicar el modelo a los datos de prueba, se obtuvo un **error cuadrático medio (MSE) de 5.95**, lo que indica un comportamiento de subajuste. Este resultado sugiere que el modelo no logra capturar de manera adecuada la no linealidad presente en la relación entre las variables predictoras y la variable objetivo, como se evidencia en la visualización gráfica correspondiente.



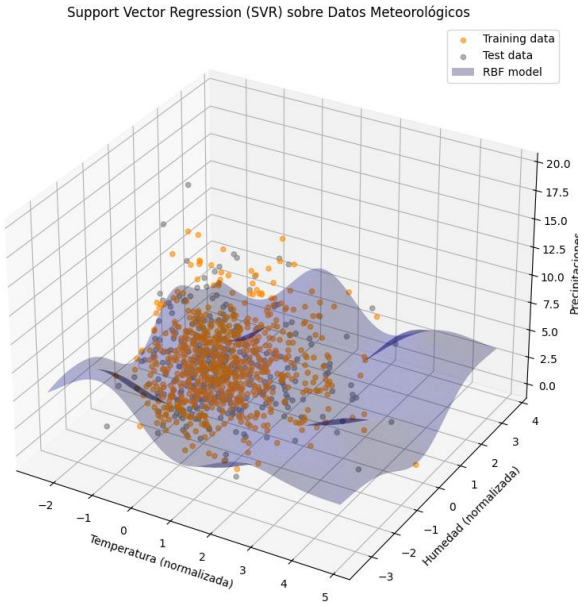
Vemos como las predicciones y los valores del modelo se agrupan en una recta y que los valores de la humedad y la temperatura están muy dispersos. Por lo tanto, el modelo no es bueno. (representación del model_SVR_1).

Para solventar este problema tomamos un kernel radial, el cual es el mejor para capturar las relaciones no lineales, además de ajustar los parámetros C y épsilon para reducir la dispersión de los datos.



El **error Cuadrático Medio** de la máquina de Soporte Vectorial con el kernel radial será de 3.99, un valor mucho mejor que el que nos sugiere la búsqueda en malla. Además, vemos en la gráfica como hemos podido solventar los problemas de subajuste del modelo y hemos podido reducir la dispersión de los datos, aunque todavía podemos encontrar ciertos valores atípicos.

A continuación, se presenta una representación tridimensional que permite visualizar con mayor claridad el grado de ajuste del modelo de Máquinas de Soporte Vectorial implementado respecto a los datos.

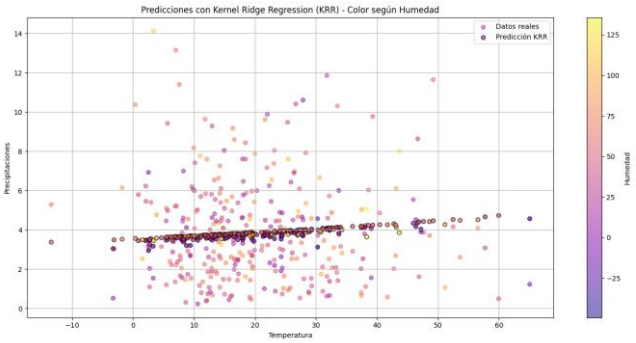


- ## Kernel Ridge Regression (KRR):

La Kernel Ridge Regression es un método de regresión que emplea un kernel al igual que SVR para poder computar modelos que se ajusten bien a los datos y puedan realizar buenas predicciones. En este caso los hiperparámetros que debemos ajustar son, además del kernel, alpha y gamma.

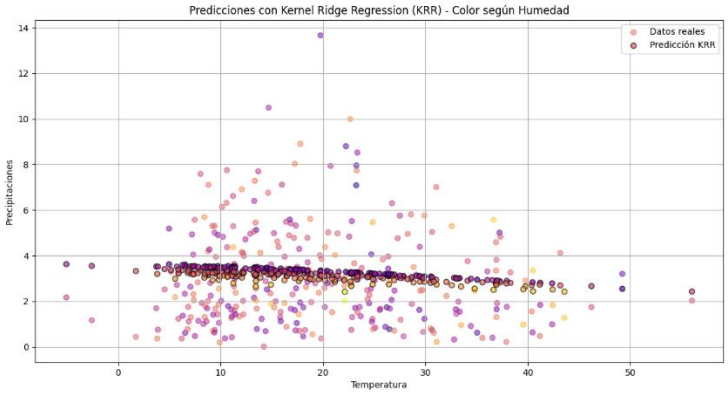
Inicialmente se seleccionó un kernel radial y unos valores de 0.3,5 para alpha y gamma. Pero para estos parámetros el modelo se sobreajustaba, y por lo tanto tras implementar la búsqueda en malla (GridSearch) y estudiar las representaciones de los parámetros implementé un modelo con un kernel sigmoidal, y unos valores de alpha = 1 para suavizar los valores muy grandes del modelo y gamma = 0.001 para reducir mucho la influencia de un solo ejemplo y tomar el conjunto de los datos de humedad y temperatura.

Obteniendo la siguiente representación gráfica:



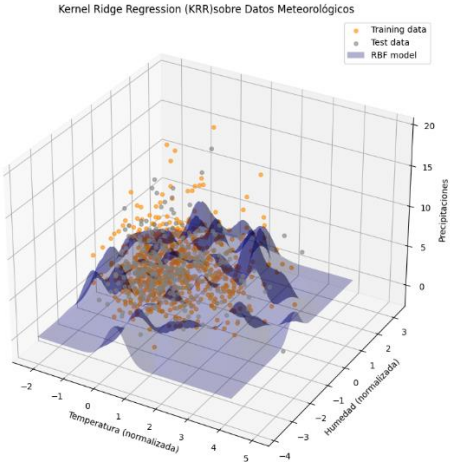
Vemos como el modelo captura mejor la variabilidad de los datos, pero no deja de ser un modelo que no es bueno con un error cuadrático medio de 11.2318, bastante elevado.

Es por ello que volví a implementar un kernel radial, pero modifiqué los parámetros α y gamma para poder ajustar mejor el modelo a los datos. Tomando un alpha = 0.2 y un gamma = 0.01 obtenemos un modelo con un **error cuadrático medio menor**, con un valor de **4.94** además de una representación más clara sobre las predicciones y el ajuste de nuestro modelo.



Vemos como reduciendo el valor de y aumentando el valor de gamma hemos conseguido capturar mejor la variación de los datos, reduciendo la dispersión de estos.

A continuación, se expone una representación tridimensional del modelo con el objetivo de observar, en el espacio de características, el grado de ajuste del modelo a los datos. Esta visualización permite analizar de forma más intuitiva cómo el modelo aproxima la superficie que representa la relación entre la humedad y la temperatura con las precipitaciones.



- ## Proceso Gaussiano de Regresión (PGR):

  Por último, vamos a analizar como ha sido el modelo del Proceso Gaussiano de Regresión. Como sabemos en este tipo de modelos que emplean procesos gaussianos para realizar las predicciones del modelo y que tienen por detrás una arquitectura basada en el Teorema de Bayes es muy importante el kernel. La elección del kernel será crítica para determinar la viabilidad de nuestro modelo.

  En este caso necesitamos un kernel compuesto, pues por la naturaleza de los datos, que son series temporales. Estos por norma suelen presentar cierta periodicidad o estacionalidad. En la gráfica de los modelos con los datos originales ya hemos visto que existe esta periodicidad en los datos.

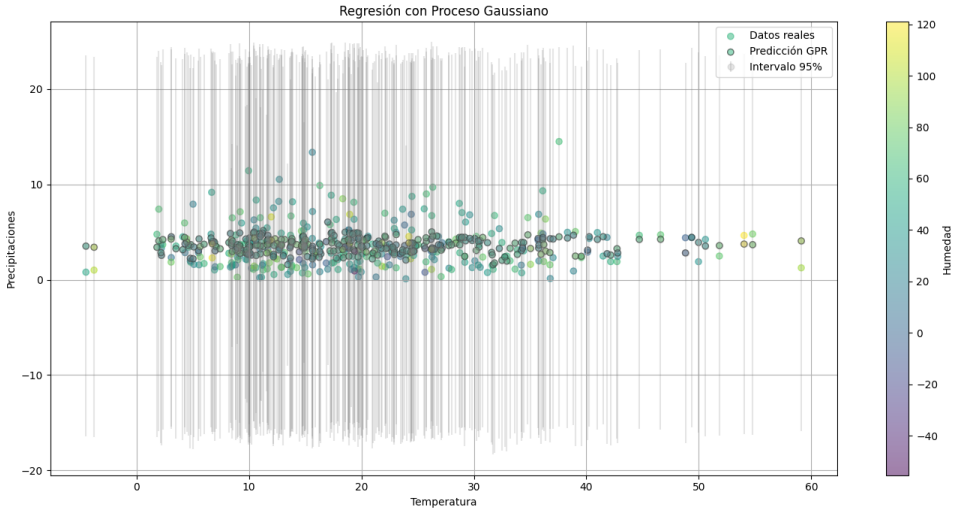
  Es por ello que el kernel tendrá una componente para capturar la tendencia lineal (DotProduct), una componente para capturar la periodicidad (ExpSineSquared) y una última para capturar las variaciones suaves, es decir las relaciones no lineales de los datos (RBF).

  Implementando estas componentes obtenemos el siguiente kernel compuesto:

```
kernel_2 = C(1.0, (1e-2, 1e2)) * RBF(0.5, (1e-2, 1e2)) + ExpSineSquared(length_scale=1.0,
                                                                         periodicity=1.0,
                                                                         length_scale_bounds=(0.1, 10.0),
                                                                         periodicity_bounds=(0.5, 1.5))
```

  Además de este kernel debemos ajustar los parámetros alpha y n_restart_optimizer que se corresponde al número de veces que se reinicia el modelo.

  Tras ajustar estos datos obtenemos la siguiente representación gráfica:



Posee un **error cuadrático medio de 6.0118**. El error cuadrático medio es muy bueno, pues la escala del eje “y” va desde -20 a 20 aproximadamente, lo cual nos permite afirmar que las predicciones que realiza el modelo son bastante buenas. Además, en cuanto al modelo representado todos los valores se encuentran dentro del intervalo de confianza que hemos impuesto.

Los puntos correspondientes a los valores reales son muy cercanos a la línea creada por las predicciones del modelo, que, aunque no la haya dibujado se observa que se encuentra entre 0 y 10 con un valor aproximado de 4.

Comparación de los modelos:

Tras el análisis comparativo de los tres modelos de aprendizaje automático implementados para la predicción de precipitaciones en función de la temperatura y la humedad, podemos concluir que el modelo **(SVR)** ofrece el mejor desempeño en términos de precisión. Este modelo presenta el **menor error cuadrático medio (MSE)**, con un valor de **3.99**, superando al modelo de **(PGR)** y **(KRR)**, cuyos errores cuadráticos medios fueron de **6.0118** y **4.94**, respectivamente. Estos resultados indican que el modelo SVR logra una mejor aproximación a los valores reales de precipitación en el conjunto de datos evaluado.

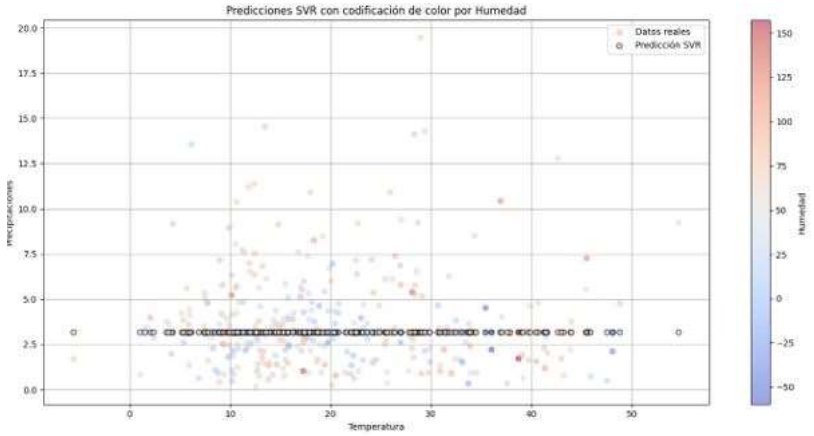
Es evidente que la diferencia del valor de los **errores cuadráticos medios** es muy pequeña, todos son bajos y relativamente parecidos. Pero, si tuviese que centrarme en la representación gráfica de un modelo para poder ver si el modelo se ajusta bien optaría por escoger la representación de los **(PGR)**, pues creo que pese a que posee un error cuadrático más elevado su representación es mas clara y consigue eliminar los valores atípicos, pues todos se encuentran dentro del intervalo de confianza definido.

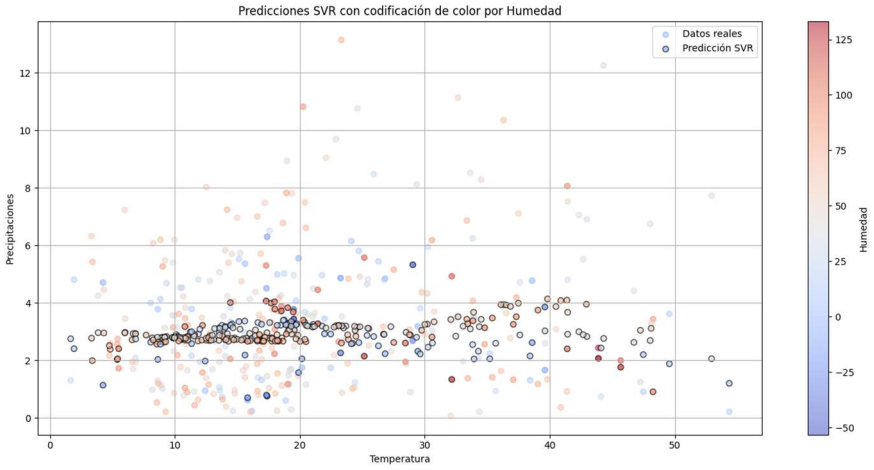
Hay peculiaridades de cada modelo, como he mencionado **PGR** consigue capturar todos los outliers o valores atípicos, pero **SVR** y **KRR** consiguen capturar los datos en un rango más pequeño reduciendo la dispersión de estos. **PGR** posee datos de las precipitaciones entre 0 y 10 a diferencia de los otros dos modelos que poseen la mayor parte de sus datos entre 2 y 6.

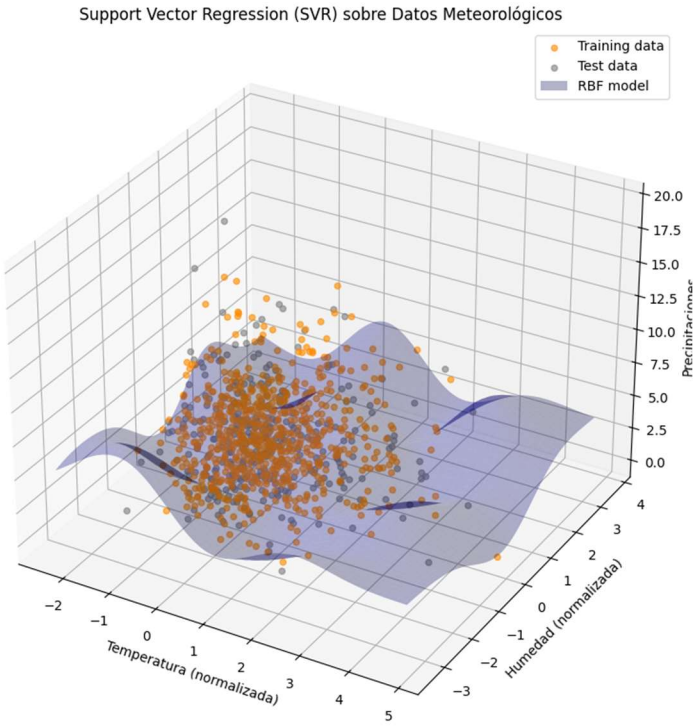
Pero cuando hablamos de capturar las relaciones no lineales de los datos podemos decir que **SVR** es el modelo que mejor las captura.

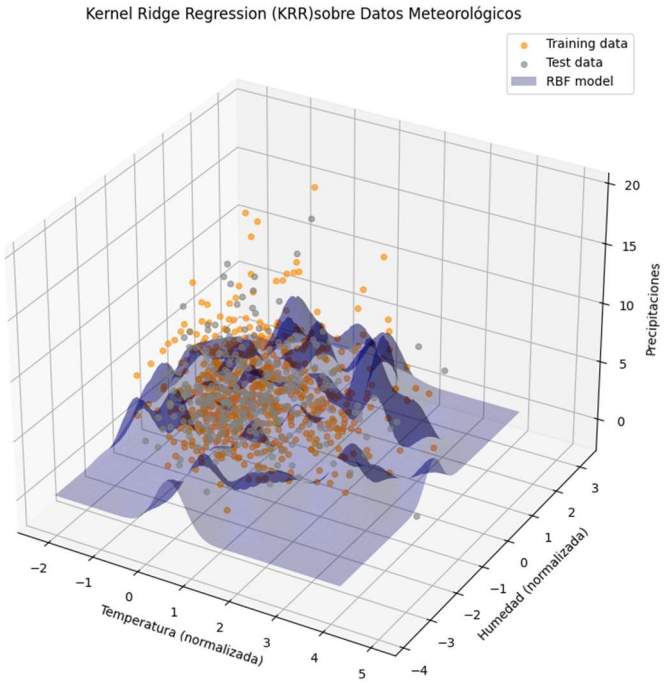
Conclusión

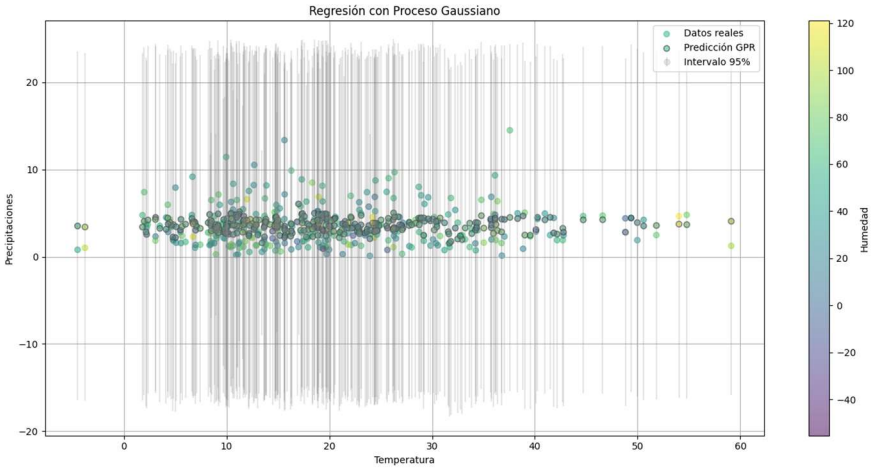
Cada modelo posee unas fortalezas distintas, pero basándome en el **error cuadrático medio** y en la forma de capturar las relaciones lineales pienso que el modelo de **Máquinas de Soporte Vectorial (SVR)** es el más adecuado para predecir las precipitaciones en función de la humedad y la temperatura.


Predicciones SVR con codificación de color por Humedad
Datos reales
Predicción SVR
Precipitaciones
Temperatura
Humedad


Predicciones SVR con codificación de color por Humedad
Datos reales
Predicción SVR
Precipitaciones
Temperatura
Humedad


Support Vector Regression (SVR) sobre Datos Meteorológicos
Training data
Test data
RBF model
Temperatura (normalizada)
Humedad (normalizada)


Kernel Ridge Regression (KRR)sobre Datos Meteorológicos
Training data
Test data
RBF model
Precipitaciones
Temperatura (normalizada)
Humedad (normalizada)


Regresión con Proceso Gaussiano
Datos reales
Predicción GPR
Intervalo 95%
Precipitaciones
Temperatura
Humedad