Taller: Problema de regresión con SVR

MLS. Machine learning supervisado
Semana 5, Taller: *SVR*
Profesor: *Fernando Lozano* - **Autor Notebook:** *Mónica Gantiva*

Introducción

Descripción

Este *jupyter notebook* contiene el material necesario para el desarrollo del Taller calificable de la Semana 5 del curso *MLS: Machine learning supervisado*. En esta tarea se presentará un problema de regresión que será resuelto aplicando SVR. Usted deberá modificar hiperparámetros del modelo para mejorar su desempeño y analizar los resultados.

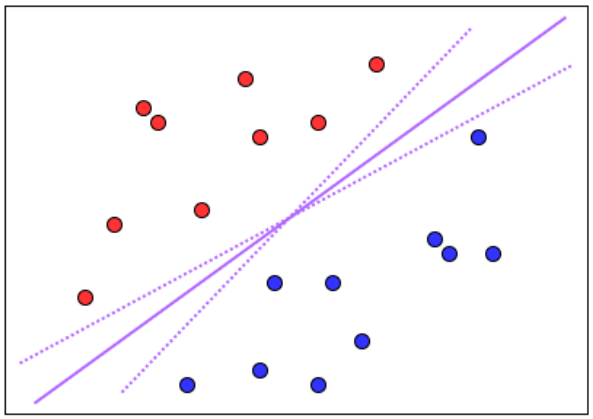
Objetivos de Aprendizaje

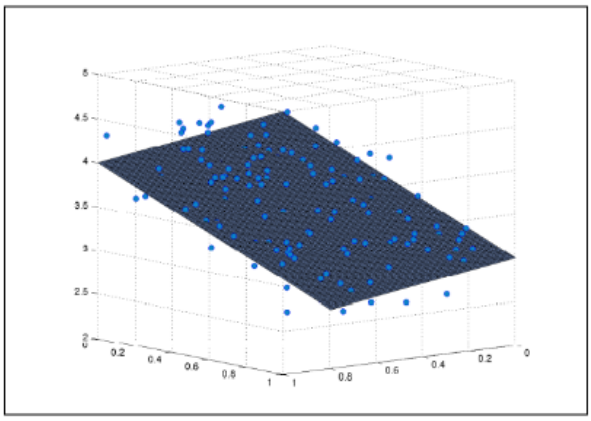
Aplicar modelos basados en kernel para resolver problemas de regresión.

Teoría

Support Vector Machine (SVM):

En terminos generales una máquina de soporte vectorial (o SVM por sus siglas en inglés) es un modelo de clasificación que busca encontrar el híperplano que separe de mejor manera los datos de dos o varias clases. Esta tarea se puede plantear como un simple problema de optimización con restricciones. En últimas lo que se desea es encontrar la separación óptima de los datos.





Un Support Vector Machine (SVM) es un algoritmo de aprendizaje supervisado que puede utilizarse tanto para clasificación como para regresión. Cuando se utiliza para regresión, se denomina Support Vector Regression (SVR). Ambos SVM y SVR comparten el mismo principio fundamental, que es encontrar un hiperplano en un espacio de alta dimensión que tenga la máxima distancia a los puntos de datos de las clases o, en el caso de SVR, que tenga la máxima distancia a los puntos de datos en la regresión.

Support Vector Regression (SVR):

Objetivo de la Regresión:

A diferencia de la clasificación, donde el objetivo es encontrar un hiperplano que separe las clases, en la regresión (SVR), el objetivo es encontrar un hiperplano que maximice la cantidad de puntos de datos (muestras) que están dentro de un margen (llamado "margen insensitivo") alrededor de la línea de regresión.

Margen Insensitivo: En SVR, se permite que algunos puntos estén dentro del margen insensitivo o incluso en el lado incorrecto del hiperplano, pero se penalizan según cuán lejos estén de la predicción.

Kernel Trick: Similar a SVM, SVR puede hacer uso del "kernel trick". El kernel transforma los datos de entrada a un espacio de mayor dimensión, permitiendo a SVR manejar relaciones no lineales entre características.

Parámetros: Algunos de los parámetros clave de SVR incluyen el tipo de kernel (lineal, RBF, polinómico, etc.), el parámetro de regularización C, y otros específicos del kernel (como el coeficiente gamma para el kernel RBF).

Metodología

En este taller se abordará un poblema aplicado de regresión para el cual usted deberá tomar el modelo de SVR y mejorar su desempeño. Para ello emplee las celdas que tienen la siguiente notación.

In [ ]:
```
# ================================================
# COMPLETAR ======================================
#

# ================================================
```

Así mismo, usted deberá análizar los resultados obtenidos para cada modelo. Para ello, responda las preguntas que se presentarán durante el taller. Dispondra de celdas con la siguiente notación para escribir su análisis:

Respuesta:

Problema aplicado: Predicción del precios de casas en Boston

Inicialización

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

Cargamos los datos

In [3]:
```
boston = load_boston()

# DataFrame de Características
data = pd.DataFrame(data=boston['data'], columns=boston['feature_names
data.head(5)
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5

Esta base de datos proviene del conjunto de datos Boston Housing Prices y contiene información de precios de viviendas en Boston

Separación de datos

In [4]:
```
X = boston.data  # características
y = boston.target  # etiquetas


# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

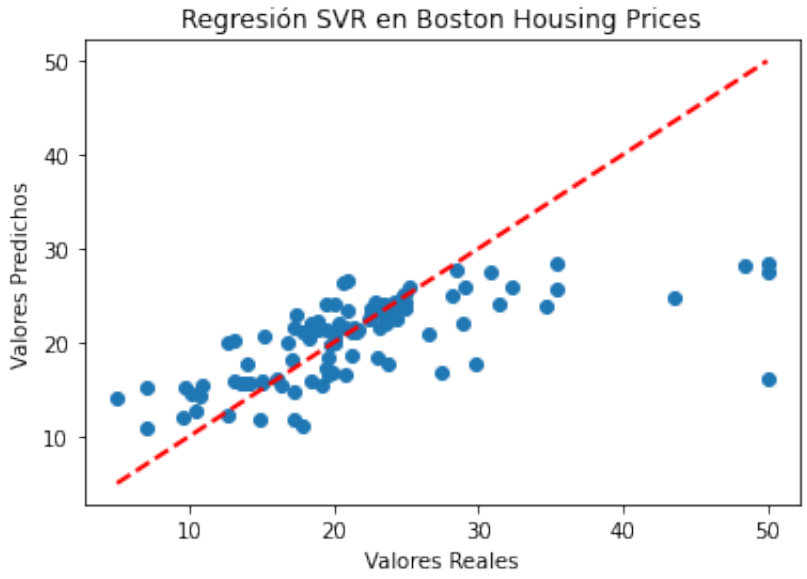
In [5]:
```
import matplotlib.pyplot as plt
```

Ejemplo: SVR

Ahora se muestra un ejemplo de cómo aplicar un SVR para realizar la predicción del precio de las casas en Boston. Observe que se crea el modelo, se hace el entrenamiento y se evalúa el desempeño del modelo para el conjunto de entrenamiento y prueba. Por último, se obtiene el desempeño del modelo. Se muestra el error de predicción y la gráfica de dispersión de valores reales frente a los valores predichos.

In [6]:
```
# Creación del modelo SVR
svr = SVR(kernel='rbf', C=10.0, epsilon=0.2)

#Entrenamiento del modelo
svr.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = svr.predict(X_test)

# Evaluar el rendimiento del modelo
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Graficar resultados
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], lines
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Regresión SVR en Boston Housing Prices')
plt.show()
```

```
Mean Squared Error: 44.33500513460085
```



Tomando como guía el ejemplo anterior, modifique los hiperparámetros del modelo con el propósito de mejorar su desempeño. Recuerde que puede modificar los siguientes parámetros:

C (parámetro de regularización): Controla el equilibrio entre un límite suave y un límite ajustado. Valores más altos de C conducen a límites más ajustados.

epsilon (margen insensitivo): Especifica la cantidad de error tolerado en la predicción. Valores más altos permiten un mayor error en las predicciones.

kernel (núcleo): Puedes experimentar con diferentes funciones de kernel, como 'linear', 'rbf' (radial basis function), 'poly' (polinómico), etc.

Varie estos hiperparámetros de la siguiente manera:

C de 0.1 a 10

epsilon de 0.1 a 0.5

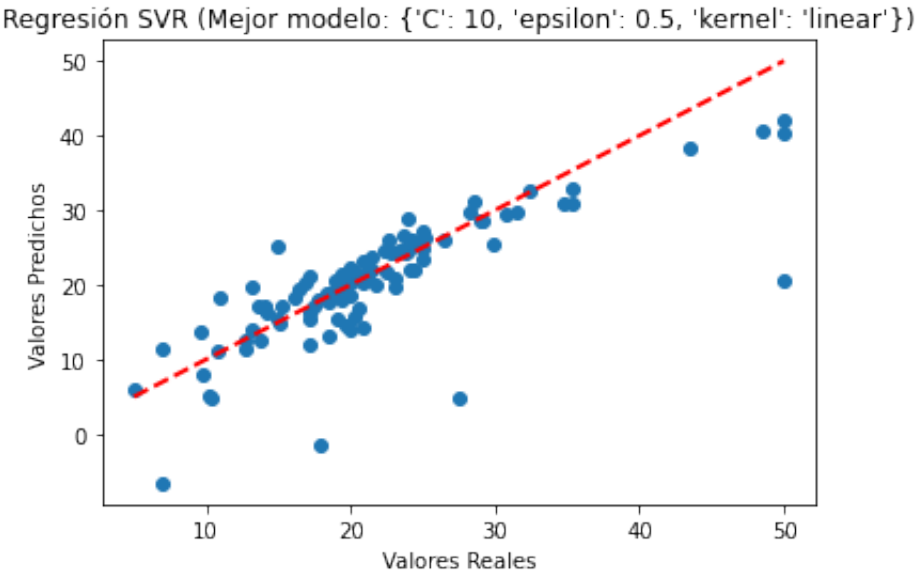
kernel, aplique los kernal:linear, rbf o poly

Pruebe diferentes combinaciones de hiperparámetros y muestre el desempeño del modelo para al menos 4 casos (el error y la gráfica de dispersión de valores reales frente a los valores predichos). *Puede utilizar GridSearchCV*

In [20]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Definir el modelo SVR y los hiperparámetros a probar
svr = SVR()
param_grid = {
    'C': [0.1, 1, 10],
    'epsilon': [0.1, 0.3, 0.5],
    'kernel': ['linear', 'rbf', 'poly']
}

# Uso de GridSearchCV para probar diferentes combinaciones de hiperpar
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5,

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

# Mostrar los mejores hiperparámetros
index_best = grid_search.cv_results_['params'].index(grid_search.best_
print(f"Mejores hiperparámetros: {grid_search.best_params_}, MSE (grid
```

```
# Evaluar el rendimiento del mejor modelo en el conjunto de prueba
best_svr = grid_search.best_estimator_
y_pred = best_svr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (test): {mse}")

# Graficar resultados para el mejor modelo
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], lines
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title(f'Regresión SVR (Mejor modelo: {grid_search.best_params_})')
plt.show()
```

```
Mejores hiperparámetros: {'C': 10, 'epsilon': 0.5, 'kernel': 'linea
r'}, MSE (grid_search): 26.35860781431718
Mean Squared Error (test): 29.33910853900462
```



In [34]:
```
results = grid_search.cv_results_

for i, param in enumerate(results['params']):
    print(f"Combinación {i + 1}: {params}, MSE (grid_search): {-resul
```

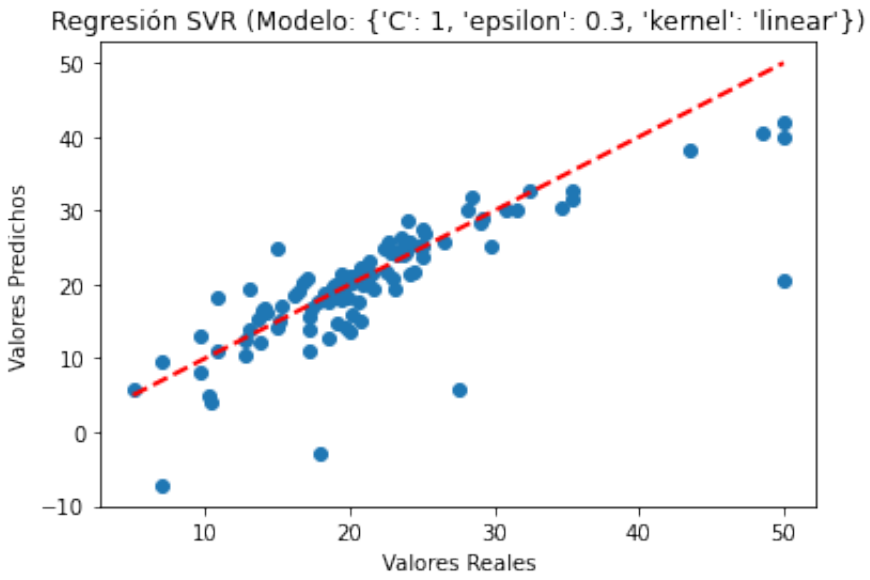
```
Combinación 1: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 26.832352714912776
Combinación 2: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 79.23446265029324
Combinación 3: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 75.52176081715662
Combinación 4: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 26.906234943278513
Combinación 5: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
```

```
earch): 79.05005055902402
Combinación 6: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 75.66409349808896
Combinación 7: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 27.042565005845358
Combinación 8: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 79.26334730302474
Combinación 9: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_s
earch): 75.41148443227073
Combinación 10: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.803620457588693
Combinación 11: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.41779067430596
Combinación 12: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.71018744393966
Combinación 13: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.71758565017985
Combinación 14: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.66497467255456
Combinación 15: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.73575883036438
Combinación 16: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.722654904983177
Combinación 17: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.78603681240119
Combinación 18: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.5631840653254
Combinación 19: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.76292748385768
Combinación 20: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 66.30319167334775
Combinación 21: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 63.32269457732527
Combinación 22: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.812965958649507
Combinación 23: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 66.28834447470254
Combinación 24: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 63.254898324664126
Combinación 25: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 26.35860781431718
Combinación 26: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 66.55141568930426
Combinación 27: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 63.12081409569436
```

In [33]:
```
import random

# Evaluación de diferentes combinaciones de hiperparámetros
```

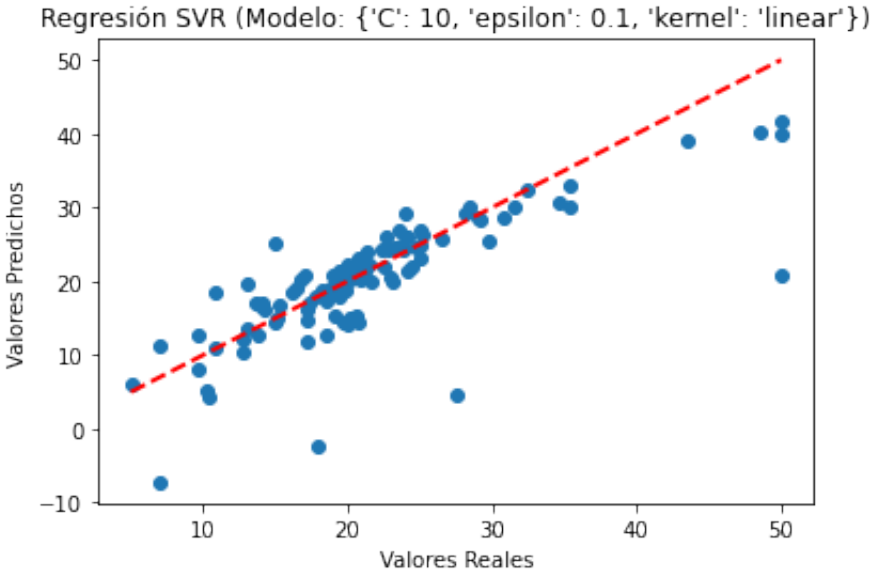
```
results = grid_search.cv_results_
mi_lista = list(range(1, 26))
elementos_aleatorios = random.sample(mi_lista, 4)

# Mostrar los MSE para al menos 4 combinaciones de hiperparámetros
for i in elementos_aleatorios:
    params = results['params'][i]
    print(f"Combinación {i + 1}: {params}, MSE (grid_search): {-result
    svr = SVR(**params)
    #Entrenamiento del modelo
    svr.fit(X_train, y_train)

    # Realizar predicciones en el conjunto de prueba
    y_pred = svr.predict(X_test)

    # Evaluar el rendimiento del modelo
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error (test) : {mse}")

    # Graficar resultados
    plt.scatter(y_test, y_pred)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], l
    plt.xlabel('Valores Reales')
    plt.ylabel('Valores Predichos')
    plt.title('Regresión SVR en Boston Housing Prices')
    plt.title(f'Regresión SVR (Modelo: {params})')
    plt.show()
```

```
Combinación 13: {'C': 1, 'epsilon': 0.3, 'kernel': 'linear'}, MSE (gr
id_search): 26.71758565017985
Mean Squared Error (test) : 29.59826218249995
```

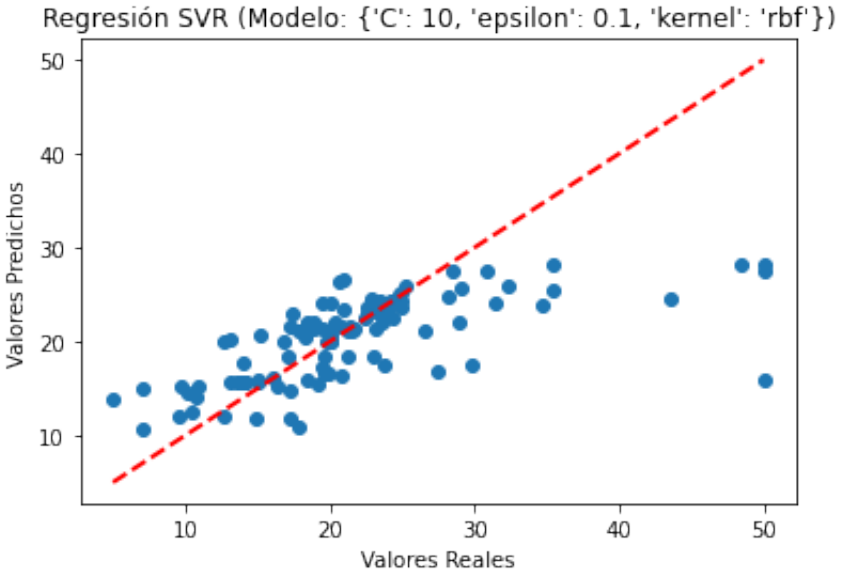



```
Combinación 19: {'C': 10, 'epsilon': 0.1, 'kernel': 'linear'}, MSE (g
```

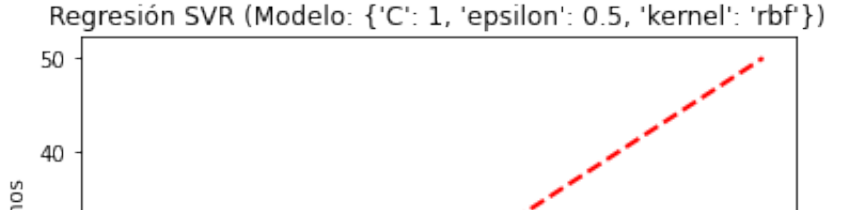
```
rid_search): 26.76292748385768
Mean Squared Error (test) : 30.35672664049693
```

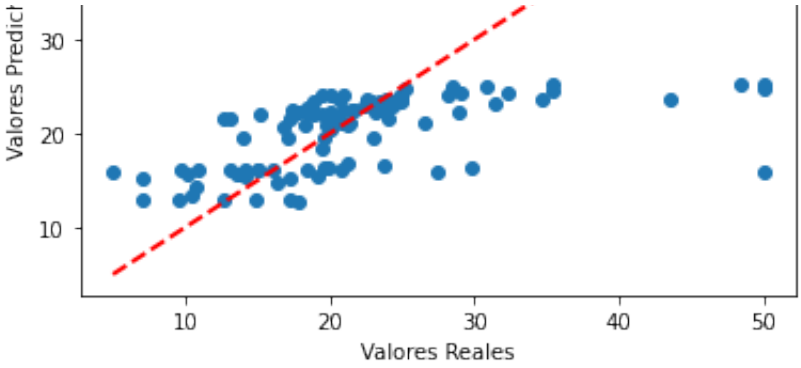



```
Combinación 20: {'C': 10, 'epsilon': 0.1, 'kernel': 'rbf'}, MSE (grid
_search): 66.30319167334775
Mean Squared Error (test) : 44.591541213219294
```




```
Combinación 17: {'C': 1, 'epsilon': 0.5, 'kernel': 'rbf'}, MSE (grid_
search): 71.78603681240119
Mean Squared Error (test) : 52.75960380236299
```





Responda la siguientes preguntas:

1. ¿Cómo afecta la variación de estos hiparámetros en el desempeño del modelo? Justifique su respuesta para el C, los diferentes kernel empleado y epsilon
2. ¿Cuál fué el modelo con mejor desempeño? ¿Cómo se refleja esto en la gráfica de dispersión?

RESPUESTAS

1. ¿Cómo afecta la variación de estos hiperparámetros en el desempeño del modelo? Justifique su respuesta para el C, los diferentes kernel empleados y epsilon.

La variación de los hiperparámetros afectó significativamente el desempeño del modelo SVR: C (parámetro de regularización):

Se probaron valores de 0.1, 1 y 10. Un C mayor (10) generalmente produjo mejores resultados (menor MSE). Esto sugiere que el modelo se benefició de un límite más ajustado, permitiendo una mayor complejidad para capturar mejor los patrones en los datos.

Kernel:

Se probaron kernels 'linear', 'rbf' y 'poly'. El kernel lineal mostró el mejor desempeño en general, con los MSE más bajos. Los kernels 'rbf' tuvieron un desempeño significativamente peor, con MSE mucho más altos. Esto indica que la relación entre las características y el precio de las casas en este conjunto de datos puede ser aproximadamente lineal.

Epsilon:

Se probaron valores de 0.1, 0.3 y 0.5. Valores más altos de epsilon (0.5) tendieron a producir mejores resultados. Esto sugiere que permitir un margen de error más amplio en las predicciones ayudó al modelo a generalizar mejor, evitando el sobreajuste.

2. ¿Cuál fue el modelo con mejor desempeño? ¿Cómo se refleja esto en la gráfica de dispersión?

El modelo con mejor desempeño fue: C = 10, epsilon = 0.5, kernel = 'linear' MSE (grid_search): 26.35860781431718 MSE (test): 29.33910853900462 Este modelo se refleja en la gráfica de dispersión de la siguiente manera:

Los puntos están más cerca de la línea diagonal ideal (y=x), lo que indica predicciones más precisas. Hay menos dispersión de los puntos alrededor de la línea diagonal. La nube de puntos sigue más de cerca la tendencia lineal, lo que es coherente con el kernel lineal elegido. Hay menos valores atípicos o predicciones extremas que se desvíen significativamente de los valores reales.

En comparación, los modelos con peor desempeño (especialmente los que usaron kernel 'rbf') muestran gráficas de dispersión con puntos más alejados de la línea diagonal y una mayor di

In [ ]: