Implementación de Regresión Lineal Múltiple y Análisis de componentes principales (PCA) en los precios medios de las viviendas en la ciudad de Boston, Estados Unidos.

Juan Esteban Jiménez Daza[1]

[1]Departamento de Matemática y Estadística, Universidad Nacional De Colombia, Manizales, Colombia.
[1]Facultad de Ciencias Exactas y Naturales.

5/06/2023

Resumen

En este estudio, se investiga la implementación de dos técnicas de análisis de datos, la regresión lineal múltiple y el análisis de componentes principales (PCA), aplicadas a los precios medios de las viviendas en Boston. El objetivo principal es analizar la relación entre múltiples variables predictoras y el precio de las viviendas, así como reducir la dimensionalidad del conjunto de datos utilizando PCA.

Se utilizó un conjunto de datos que incluye diversas características de las viviendas en Boston, como el número de habitaciones, la tasa de criminalidad en el vecindario, la proximidad al centro de la ciudad, entre otros, que se muestran en el análisis. Se aplicó la regresión lineal múltiple para modelar la relación entre estas variables predictoras y el precio de las viviendas.

1. Introducción

El análisis de datos se ha convertido en una herramienta fundamental en diversos campos, incluido el sector inmobiliario. La comprensión de los factores que influyen en los precios de las viviendas es crucial para los profesionales del sector, ya que les permite tomar decisiones informadas sobre inversión, valoración de propiedades y desarrollo de estrategias de mercado. En este contexto, la implementación de técnicas de análisis como la regresión lineal múltiple y el análisis de componentes principales (PCA) puede proporcionar una visión más profunda y precisa de la relación entre las variables predictoras y el precio de las viviendas.

Se toma el estudio **'The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics Management**

publicado en 1978. Con lo cual, con los datos suministrados, se realiza una adaptación en el lenguaje de programación **Python** usando las técnicas tomadas del álgebra lineal y estadística, como la Regresión Lineal, Regresión Múltiple, Análisis de componentes principales (PCA), Error cuadrático medio, coeficiente de determinación (R2) y la Validación cruzada aplicada en ambos métodos.

2. Metodología

La metodología aplicada en este trabajo, fue la implementación de diferentes modelos del álgebra lineal para la proyección, cálculo y la graficación de matrices creadas a partir del modelo de ***Mínimos cuadrados*** y a su vez, implementadas a modelos estadísticos como lo son ***Coeficiente de determinación y error cuadrático medio.*** También modelos de machine learning como lo es ***Validación cruzada*** explicadas mas adelante.

Antes que nada, hay plantear todos nuestros modelos y transformar la base de datos seleccionada y moldearla en el lenguaje de programación **Python**, ayudándonos de las diferentes librerías para la interpretación de datos, como lo son **Pandas** para el manejo e interpretación de los datos, **Matplotlib** para la graficacíon de los resultados, algunas funciones de **Sklearn** para entrenamiento de datos y algunas propias de regresión lineal para la comparación del modelo creado con la librería.

2.1. Lectura de datos

Para la lectura de datos, primero que nada se descarga la base de datos, en este caso se usa el formato de **CSV** de excel para un mejor manejo de los datos usando Pandas, en código nos queda de la siguiente manera:

```python
from sklearn.model_selection import train_test_split

### LECTURA DE DATOS DEL CSV
boston = pd.read_csv("/content/drive/MyDrive/PROGRAMACION_II/ProyectoFinal/boston.csv")
datos = pd.DataFrame(boston)
datos.dropna(inplace = True) # LIMPIAMOS LOS DATOS CON VALOR 'NaN'

#### DEFINIMOS LAS VARIABLES X and Y ####
# Usando 'drop' de pandas, seleccionamos del dataframes todos los datos y eliminamos el que no necesitemos. Con axis = 1 para eliminar la columna
x = datos.drop(['MEDV'], axis = 1).values
y = datos['MEDV'].values

### DIVISION DE DATOS. ENTRENAMIENTO/PRUEBA ###
# RELACION 80/20 #
x_entrenamiento, x_prueba, y_entrenamiento, y_prueba = train_test_split(x, y, test_size=0.20, random_state=42)
```

Se puede observar que importamos el archivo **CSV** directamente desde Drive, en donde tenemos nuestra base de datos con 507 datos, de los cuales usaremos la proporción **80/20**, siguiente a esto convertimos nuestros datos en un ***DataFrame*** para mostrar los datos directamente en una tabla mucho mas legible. Se muestra de la siguiente forma:

```
1 boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Podemos observar los siguientes datos mostrados en el estudio, detallados a continuación:

- **CRIM:** Tasa de criminalidad per cápita por barrio.
- **ZN:** Proporción de terrenos residenciales zonificados para lotes de más de 25,000 pies cuadrados.
- **INDUS:** Proporción de acres de negocios no minoristas por ciudad.
- **CHAS:** Variable del rio Charles (1 si el área limita con el río; 0 en caso contrario).
- **NOX:** Concentración de óxidos de nitrógeno (partes por cada 10 millones).
- **RM:** Numero de habitaciones de la vivienda.
- **AGE:** Tiempo en el que ha estado ocupado la vivienda desde su construccion.
- **DIS:** Distancia desde la vivienda hasta los centros de trabajo de Boston.
- **RAD:** Índice de accesibilidad a carreteras radiales.
- **TAX:** Tasa de impuesto a la propiedad de valor total por cada 10,000 dolares.
- **PTRATIO:** Relación alumno-maestro por ciudad.
- **B:** El resultado de la ecuación $B = 1000(Bk - 0{,}63)^2$ donde Bk es la proporción de personas negras por ciudad.
- **LSTAT:** porcentaje de estatus socio económico bajo de la población.
- ***MEDV:*** Valor mediano de las viviendas ocupadas por el propietario en miles de dólares. A su vez es la variable objetivo a predecir o **'target'**.

2.2. Regresión Lineal Múltiple

La regresión lineal múltiple es una técnica estadística utilizada para predecir o modelar la relación entre una variable dependiente y dos o más variables independientes. A diferencia de la regresión lineal simple, donde solo se utiliza una variable independiente, la regresión lineal múltiple permite considerar múltiples factores simultáneamente para predecir el valor de la variable dependiente. El modelo de regresión lineal múltiple se define mediante la siguiente ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_m X_m + \epsilon \tag{1}$$

Donde:

- Y es la variable dependiente que se quiere predecir.
- $\beta_0, \beta_1, \beta_2, ..., \beta_m$ son los coeficientes de regresión que representan el efecto de cada variable independiente en la variable dependiente.
- $X_1, X_2, ..., X_m$ son las variables independientes que se utilizan para predecir Y.
- ϵ es el término de error, que representa la variabilidad no explicada por el modelo.

Entonces podemos reformular el problema como una sistema de ecuaciones lineales.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_m X_m + \epsilon \tag{2}$$

$$\iff \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \tag{3}$$

$$\iff Y = A\beta \tag{4}$$

Usando mínimos cuadrados, la solución del problema esta dada por:

$$\beta = (A^T A)^{-1} A^T Y \tag{5}$$

Esto siempre y cuando la matriz $(A^T A)^{-1}$ exista.
Implementado en código, quedaría con los siguientes métodos:

- $\beta_1, \beta_2, ..., \beta_m =$

```
### CREAMOS BETA DE LA FORMULA CON MINIMOS CUADRADOS ###
def Minimos_Cuadrados(self):
    X = self.Matriz_X()
    X_tran = np.transpose(X)
    beta_hat = np.linalg.inv(X_tran @ X) @ X_tran @ self.y
    return beta_hat
```

- Matriz con 1 y valores $x_1, x_2, .., x_n =$

```python
#### CREAMOS LA MATRIZ DE DATOS X ####
def Matriz_X(self):
  fil, col = self.fil, self.col
  X = np.ones((fil, col+1))  # Se crea una matriz de 1 con dimensiones : numero de filas y numero de columnas con los datos del DataFrame 'x'
  X[:,1:] = self.x  #Colocamos los datos desde la segunda columna, hasta la ultima, La primera columna es de solo
  return X
```

- $\mathrm{Y} = \mathrm{A}\beta$

```python
### PREDICCION DE LOS DATOS 'y' NUEVOS ###
def Prediccion_y_(self):
  X = self.Matriz_X()
  beta_hat = self.Minimos_Cuadrados()
  y_predict = X @ beta_hat
  return y_predict
```

2.3. Mínimos Cuadrados

El objetivo del método de los mínimos cuadrados es minimizar el error de predicción entre los valores reales y predichos. Por ejemplo, si tienes un conjunto de puntos en un gráfico y quieres encontrar la línea que mejor se ajuste a esos puntos, puedes usar el método de mínimos cuadrados para encontrar la línea que minimiza la suma de los cuadrados de las distancias verticales entre cada punto y la línea.

Sean $A_{m\times n}$, $b \in R^n$ El problema de mínimos cuadrados consiste en encontrar $x \in R^n$, tal que:

$$||Ax - b|| \tag{6}$$

sea mínimo. Es decir, $\hat{x} \in R^n$ es una solución por mínimos cuadrados, si:

$$\forall x \in R^n, \quad ||A\hat{x} - b|| \leq ||Ax - b|| \tag{7}$$

Sea W un subespacio vectorial de R^n, $y \in R^n$,
$\hat{y} = \mathrm{Pro}y_W\, y$ la proyección ortogonal de y sobre W
Entonces, $\hat{y}$ es el vector en W más cercano a y en el sentido que

$$\forall v \in W, \quad ||y - \hat{y}|| \leq ||y - v|| \tag{8}$$

La solución al problema de mínimos cuadrados, esta dada por:

$$\hat{x} = \text{Pro}y_W b \tag{9}$$
$$= (A^T A)^{-1} A^T b \tag{10}$$

donde W es el espacio columna de la matriz A.

2.4. Error cuadrático medio (MSE)

Es una medida del error cuadrático promedio de las predicciones del modelo en comparación con los valores reales.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2 \tag{11}$$

Implementado en código, quedaría formado por este método:

```
### ERROR CUADRATICO MEDIO ###
def ErrorCuadraticoMedio(self):
  y_predict = self.Prediccion_y_()
  mse = np.mean((y_predict - self.y) ** 2)
  return mse
```

2.5. Coeficiente de determinación

El coeficiente de determinación (R cuadrado) indica la cantidad proporcional de variación en la variable de respuesta y, explicada según las variables independientes X en el modelo de regresión lineal. Cuanto mayor sea el R cuadrado, mayor será la variabilidad explicada por el modelo de regresión lineal. Por lo general, R2 oscila entre 0 y 1, donde 1 indica un ajuste perfecto.

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} = 1 - \frac{\sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{12}$$

$$\tag{13}$$

$$R^2_{ajustado} = 1 - (\frac{n-1}{n-p}) \frac{\sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2}{\sum_{i=1}^{n} (Y_i - \bar{Y})^2} \tag{14}$$

$$\tag{15}$$

Implementado en el código, quedaría formado por el método:

```python
### COEFICIENTE DE DETERMINACION R2 ###
def R2(self, y_hat, y, P = None):
  n = len(self.y)
  y_bar = np.mean(self.y) # Calcular la media de los valores reales

  if P == None:

    # Calcular la suma total de los cuadrados de las diferencias entre los valores reales y la media
    R2 = sum( (y_hat - y_bar)**2 ) / sum( (y - y_bar)**2 )
    #print(R2)
    return R2

  # R2 AJUSTADO
  else:
    num=sum((y_hat-y)**2)
    dem=sum((y-y_bar)**2)
    R2_ajustado = 1-((n-1)/(n-P))*(num/dem)
    return R2_ajustado
```

2.6. Validación cruzada

La validación cruzada en regresión es una técnica utilizada para evaluar la capacidad predictiva de un modelo de regresión. La idea es dividir el conjunto de datos en dos partes: un conjunto de entrenamiento y un conjunto de prueba. El modelo se entrena con el conjunto de entrenamiento y se evalúa con el conjunto de prueba.

Sin embargo, una única división de los datos puede llevar a resultados sesgados, ya que el modelo puede ser muy sensible a la partición de los datos. Por lo tanto, la validación cruzada se utiliza para evaluar el modelo en múltiples particiones diferentes de los datos.

- La validación cruzada en regresión se lleva a cabo de la siguiente manera:

  1. Se divide el conjunto de datos en k partes iguales (k es un número elegido por el usuario, por lo general entre 5 y 10).
  2. Se selecciona una de las partes como conjunto de prueba y se entrena el modelo con las partes restantes.
  3. Se evalúa el modelo con el conjunto de prueba.
  4. Se repiten los pasos 2 y 3 k veces, de manera que cada parte del conjunto de datos se utiliza una vez como conjunto de prueba.
  5. Se calcula la medida de ajuste (como el coeficiente de determinación) para cada una de las k iteraciones
  6. Se promedian las medidas de ajuste para obtener una medida global del ajuste del modelo.

El resultado de la validación cruzada en regresión es un valor promedio de la medida de ajuste para el modelo. Este valor es una mejor estimación del ajuste del modelo que el valor calculado a partir de una única partición de los datos.

Su implementación en el código, quedaría formado por el método:

```python
### VALIDACION CRUZADA ###
def validacion_cruzada2(self, k, P='N'):
  indices = np.random.permutation(self.fil)  # Se realiza una permutación aleatoria de los índices de los datos
  particiones = [indices[j::k] for j in range(k)]  # Se dividen los índices en k particiones
  resultados = []

  for i in range(k):
    prueba = particiones[i]
    entrenamiento = np.concatenate([particiones[j] for j in range(k) if j != i])  # Se concatenan los índices de las particiones de entrenamiento

    x_entrenamiento, y_entrenamiento = self.x[entrenamiento], self.y[entrenamiento]
    x_prueba, y_prueba = self.x[prueba], self.y[prueba]
    modelo_i = RegresionMultiple(x_entrenamiento, y_entrenamiento)  # Se crea un nuevo modelo para cada partición de entrenamiento
    y_hat_prueba = modelo_i.Evaluar(x_prueba)  # Se obtienen las predicciones en los datos de prueba

    if str(P) == 'N':
      R2 = modelo_i.R2(y_hat_prueba, y_prueba)  # Coeficiente de determinación estándar
    else:
      R2 = modelo_i.R2(y_hat_prueba, y_prueba, P)  # Coeficiente de determinación ajustado

    resultados.append(R2)

  R2_promedio = np.mean(resultados)  # Promedio de los coeficientes de determinación obtenidos en cada partición
  return R2_promedio
```

2.7. Análisis de componentes principales (PCA)

El análisis de componentes principales (PCA), también conocido como análisis de factores principales, es una técnica estadística utilizada para reducir la dimensionalidad de un conjunto de datos, al mismo tiempo que se conserva la mayor cantidad posible de información. El objetivo principal del PCA es encontrar una representación más compacta de los datos, sustituyendo las variables originales por un conjunto reducido de variables llamadas componentes principales.

- El proceso para realizar un PCA consta de los siguientes pasos:

1. Estandarizar los datos: Es común estandarizar las variables originales para que tengan media cero y desviación estándar uno. Esto es importante para asegurar que las variables con diferentes escalas no dominen el análisis.
   su formula es la siguiente:

$$Z = \frac{X - \mu}{\sigma}$$

   Implementado en el código seria de la forma:

```python
def Estandarizar(self, X):
  media = self.X.mean(axis = 0)
  desvStand = self.X.std(axis = 0)
  X_escalado = np.zeros_like(X)

  for k in range(self.columnas):
    if desvStand[k] != 0:
      X_escalado[:,k] = (X[:,k] - media[k]) / desvStand[k]
    else:
      X_escalado[:,k] = X_escalado[:,k]
  return X_escalado
```

2. Calcular la matriz de covarianza o correlación: Se calcula la matriz de covarianza si las

variables tienen unidades diferentes, o la matriz de correlación si las variables están en la misma escala. Esta matriz captura las relaciones entre las variables originales.

3. Obtener los autovectores y autovalores: Se calculan los autovectores y autovalores de la matriz de covarianza o correlación. Los autovectores representan las direcciones principales de variación en los datos y los autovalores indican la importancia relativa de cada autovector.

4. Ordenar los autovectores: Los autovectores se ordenan en función de sus autovalores, de mayor a menor. De esta manera, los primeros autovectores (componentes principales) explican la mayor parte de la variabilidad total de los datos.

```
def MatrizProyeccion(self, margen, X):
  U, S, VT = np.linalg.svd(X) # Descomposición en valores singulares
  V = VT.T # Transponemos la matriz V para que las columnas sean los vectores propios de X.T@X
  S = S**2 # Obtener los valores propios = cuadrado de los valores singulares
```

5. Seleccionar el número de componentes principales: Se selecciona el número de componentes principales que se utilizarán para la reducción de dimensionalidad. Esto puede basarse en criterios como la varianza explicada acumulada o la regla del codo.

```
# Calcular el número de componentes principales
porcVarAcum = 0
k = 1
while porcVarAcum < margen:
  porcVarAcum = sum(S[:k]) / sum(S)
  k += 1
PCA.num_comp = k

matrizPro = V[:,:PCA.num_comp]#La matriz de proyección
print("Número de componentes principales utilizadas:", PCA.num_comp)

return matrizPro
```

6. Proyectar los datos en el nuevo espacio de componentes principales: Finalmente, se proyectan los datos originales en el espacio definido por los autovectores seleccionados, obteniendo así las nuevas variables (componentes principales) que resumen la información de las variables originales.

$$Z = XW$$

```
def Proyectar(self,Datos):
  Z = Datos @ self.matrizPro
  return Z
```

3. Resultados

Los resultados mostrados, son **El coeficiente de determinación, el error cuadrático y validación cruzada con una división de los datos en 5 partes.** en código nos muestra:

```python
from sklearn import linear_model
from sklearn.metrics import r2_score, mean_squared_error

### INVOCAR A LA CLASE REGRESION LINEAL MULTIPLE ###
Regresion = RegresionMultiple(x_prueba,y_prueba)
y_predict = Regresion.Prediccion_y_()
```

```python
valid = Regresion.validacion_cruzada2(5)
print('Validacion cruzada con 5:', round(valid,4))

R2 = Regresion.R2(y_predict, y_prueba)
print('\nCoeficiente de determinacion R2:', round(R2,4))

R2_2 = r2_score(y_prueba , y_predict)
print('Coeficiente R2 sklearn: ', round(R2_2,4))

MSE = Regresion.ErrorCuadraticoMedio()
print('\nError cuadratico medio:', round(MSE,4))

mse = mse=mean_squared_error(y_prueba ,y_predict)
print('Error cuadratico sklearn:', round(mse,4))
```

```
Validacion cruzada con 5: 0.9474

Coeficiente de determinacion R2: 0.7656
Coeficiente R2 sklearn:  0.7656

Error cuadratico medio: 17.193
Error cuadratico sklearn: 17.193
```

3.1. Muestra de resultados

- En primer estancia, se crea un objeto iterable, para mostrar los resultados que nos muestra los métodos de **R2, Error cuadrático y validación cruzada**, en este caso el objeto iterable es ***Regresión***
- Importamos la librería de sklearn y algunos métodos para hacer la comparación de los resultados de la librería, con el código implementado
- Se puede apreciar tanto, que el coeficiente de determinación y el error cuadrático, con el código implementado y la librería sklearn son iguales, con lo cual el modelo esta bien modelado y retorna datos válidos.

Sin embargo, se puede apreciar que el coeficiente de determinación es del **76 %** que se interpreta como la variabilidad entre los datos de prueba y los datos de prueba predecidos. Con lo cual es un porcentaje bajo, que se traduce en que el modelo de regresión lineal múltiple en este conjunto de datos, no es el mas apropiado para la predicción; En este caso el valor medio de las viviendas en Boston.

- En la validación cruzada, se nota un coeficiente diferente al del R2, esto es debido a la cantidad de datos evaluados.

3.2. Comparación de precios reales y predecidos

```
### COMPARACION DE DATOS ORIGINALES Y PREDICHOS EN DATAFRAME ###
datos_y_pred = pd.DataFrame({'Precios reales': y_prueba, 'Precios Predecidos': y_predict, 'Diferencia': y_prueba - y_predict})
datos_y_pred[0:500]
```

	Precios reales	Precios Predecidos	Diferencia
0	23.6	28.798297	-5.198297
1	32.4	35.627105	-3.227105
2	13.6	12.618215	0.981785
3	22.8	23.722885	-0.922885
4	16.1	16.701156	-0.601156
...	...	...	...
97	17.9	9.312596	8.587404
98	9.6	16.614370	-7.014370
99	17.2	19.557134	-2.357134
100	22.5	21.369729	1.130271
101	21.4	23.216047	-1.816047

102 rows × 3 columns

Se puede observar que para el uso de regresión, se tomaron en cuenta 102 de los 507 datos en total, en la columna **Diferencia** se aprecia la variabilidad entre el precio real en los datos, y los precios predecidos por el modelo de regresión. A su vez esto explica el porque el coeficiente de determinación es de un valor bajo y la validación cruzada es muy volátil.

3.3. Análisis de componentes principales(PCA)

En el análisis de componentes principales, el algoritmo, tomó en cuenta 4 de las 13 variables independientes en los datos, Sin embargo tanto como el **Coeficiente de determinación ajustado y el error cuadrático** sus resultados son casi iguales con muy poca variabilidad respecto al modelo de regresión múltiple.

```
### INVOCAR CLASE PCA ###

Componentes_principales = PCA(x, y, 0.6)
y_predict_pca = Componentes_principales.Prediccion_y_()
```

```
Número de componentes principales utilizadas: 4
```

```
Validacion_Cruzada_pca = Componentes_principales.Validacion_Cruzada_pca(5)
print('\nValidacion cruzada con PCA: ', Validacion_Cruzada_pca)

R2_ajustado = Componentes_principales.R2(y_predict_pca,y)
print('\nCoeficiente de determinacion R2 ajustado: ', R2_ajustado)

mse_pca = Componentes_principales.ErrorCuadraticoMedio()
print('\nError cuadratico medio: ', mse_pca)
```

```
Validacion cruzada con PCA:  0.6556815749168357

Coeficiente de determinacion R2 ajustado:  0.6456592995092347

Error cuadratico medio:  29.73558198232594
```

Conclusiones

Con los resultados mostrados gracias a los modelos de regresión lineal múltiple y análisis de componentes principales, se pueden determinar los siguientes puntos:

1. Con los datos de estudio, se puede determinar que el modelo de **regresión lineal múltiple y análisis de componentes principales** no son la mejor opción para modelar la predicción del valor medio de las viviendas en Boston, por su bajo porcentaje de coeficiente de determinación, que fué del ***76 %***.

2. Explorar diferentes métodos de predicción para lograr un mayor porcentaje de coeficiente de determinación.

3. La volatilidad del resultado de **validación cruzada**, se debe a la poca cantidad de datos suministrados al modelo, este problema se resuelve, suministrando al modelo una mayor cantidad de datos al modelo.

Referencias

[1] *DataScientest*. URL: `https://datascientest.com/es/cross-validation-definicion-e-importancia`.

[2] *Decomposing signals in components (matrix factorization problems)*. URL: `https://scikit-learn.org/stable/modules/decomposition.html#decompositions`.

[3] FEDESORIANO. *Boston House Prices-Advanced Regression Techniques*. 2021. URL: `https://www.kaggle.com/datasets/fedesoriano/the-boston-houseprice-data?datasetId=1381830&sortBy=voteCount`.

[4] tarcux. *Decomposing signals in components (matrix factorization problems)*. 2018. URL: `https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/datasets/data/boston_house_prices.csv`.

[1] [3] [2] [4]