Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

TC3006C. Inteligencia artificial avanzada para la ciencia de datos
(Grupo 103)

Módulo 2 Análisis y Reporte sobre el desempeño del modelo.

Daniel Flores Rodríguez A01734184

14 de Septiembre 2022

Introducción: Los datos a usar

El dataset usado para la realización de esta actividad se basa en una situación bastante cotidiana, o por lo menos de una situación que muchos habremos experimentado o escuchado alguna vez, por esto me refiero a la cuestión del estudio, específicamente las horas de estudio y su relación con el puntaje o notas obtenidas en algun examen o proyecto. Para ser más precisos en cuanto a la descripción del dataset a usar, podemos decir que este solo cuenta con dos campos o atributos, siendo el primero de estos el número de horas que estudia un alumno (Hours) y el puntaje obtenido.

Inmediatamente podemos notar que ambas variables son están lógicamente relacionada una con la otra, ya podemos considerar que la variable independiente en este caso ‘x’ se asocia con las horas de estudio de cada alumno, mientras que la ‘y’, nuestra variable dependiente, se asocia con el puntaje obtenido, por lo que simplemente podremos usar estas dos variables para poder entrenar nuestro modelo.

Se espera que con estos datos y el modelo a usar, podamos calcular y comprobar que es lo que sucede con la relación de estas dos variables mencionadas anteriormente, lógicamente y como podemos ver en algunos datos históricos del dataset, se espera que a más horas de estudio, la tendencia de la calificación obtenida sea proporcional, es decir a que a más horas de estudio, el estudiante obtenga un puntaje bueno o alto, en este caso. A continuación podemos ver la estructura general del dataset.

	column 1	column 2
1	Hours	Scores
2	2.5	21
3	5.1	47
4	3.2	27
5	8.5	75
6	3.5	30

El dataset fue obtenido en Kaggle, el enlace para poder acceder a este, se encuentra en el siguiente link:
https://www.kaggle.com/datasets/himanshunakrani/student-study-hours

Herramientas usadas

Para algunas de las funcionalidades que se implementaron en esta actividad, se usan algunas librerías que nos ayudan a simplificar el trabajo realizado y nos ayudan a tener más precisión en cuanto a nuestros resultados, a continuación se mencionan algunas de las más relevantes que se usaron para la realización del modelo:

- Scikit-Learn: Librería de python dedicada al análisis de datos y aprendizaje máquina que va desde modelos simples como regresiones lineales o logísticas hasta redes neuronales complejas.

- Matplotlib: Librería de python dedicada a la graficación de datos, fundamental para poder comprender el comportamiento de los datos, por ejemplo, a partir de su ordenamiento o introducción a un modelo determinado.

- Pandas: Librería de python open source dedicada al análisis de datos de manera intuitiva y flexible. Pandas puede abarcar la gran mayoría de ETL, sobre todo siendo fundamental para la extracción, limpieza y transformación de datos a emplear.

- Numpy: Librería de python dedicada a ciencia de datos desde la perspectiva de vectorización, indexado, funciones matemáticas, funciones algebraicas, permutación de matrices, cálculo vectorial, etcétera.

Limpieza de datos

Para este caso, no se empleó algún tipo de limpieza de datos, ya en el caso de este dataset, al ser solo dos variables, y no tener algún dato vacío, se descarta automáticamente este proceso. Sin embargo, se está consciente que en caso de tener algún otro dataset con más cantidad de datos y sea difícil su visualización a simple vista, lo que se tendría que realizar es limpiar el dataset de tal manera que no tengamos datos vacíos o con cifras que puedan afectar el resultado de nuestro modelo (números negativos, cifras inusuales, entre otras) . Así mismo podríamos

decir que en el caso hipotético en el que hubiera más columnas, se tendría que elegir un conjunto de datos correlacionados y por lo tanto, se tendría que hacer un descarte del resto de columnas, y por último pero no menos importante, realizar la conversión de strings a enteros, de valores enteros a flotantes o viceversa.

Modelo de regresión lineal

Como se mencionó anteriormente, para esta actividad, se usará el modelo de regresión lineal simple, la cual consta de una variable dependiente y otra independiente. se podría decir que la base principal de una regresión lineal recae en la siguiente fórmula $y = mx + b$, de donde podemos mencionar la pendiente 'm' y la intercepción de 'b' como uno de los elementos más relevantes en la regresión lineal simple y los cuales calcularemos.
El primer paso que se implementó en el código de python fue desglosar el data frame mencionado anteriormente en dos arreglos: el arreglo del eje X con los valores de las horas estudiadas por los alumnos; y el arreglo del eje Y la cual contiene los valores de la columna de puntaje obtenidos. En este caso de se tuvo que acomodar el array de tal manera que fuera un array 2D, el cual es aceptado por las funciones de scikit-learn, esto se hizo a través de la función .reshape() .

```
28   #we asigned our x and y values, being x our independent variable (hours of study) and y our dependent variable (marks)
29   #we use reshape function in order to redimention our y and x variable arrays
30
31   X = np.array(data.iloc[:,0]).reshape(-1,1)
32   Y = np.array(data.iloc[:,1]).reshape(-1,1)
33
```

Posteriormente al haber obtenido X y Y, se realiza la división de datos en los que corresponden al test y al train con ayuda de la función train_data_split, la cual retorna 4 variables de listas divididas X y Y tanto para el set de testeo como el set de training. Esta división queda de la siguiente manera en el programa:

```
#building the model

# in order to calculate linear regresion, we need to split the data into trining and testing d
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size= 0.25, random_state = 45)

regr = LinearRegression()
regr.fit(x_train,y_train)
```

Posteriormente, como se ve en la imagen de arriba, podemos ver que se llama la función de LinearRegression() correspondiente a la librería de scikit-learn en su configuración determinada.
Ya teniendo instanciada la instancia, se adapta el modelo, con ayuda de la función .fit(), para posteriormente calcular el coeficiente resultante de la regresión lineal y también la intercepción de la pendiente.

```
#exploring our results

print("Score: ")
print(regr.score(x_test,y_test))

print("coefficient: ")
print(regr.coef_)

print("Intercept: ")
print(regr.intercept_)
```

Posteriormente al tener nuestro modelo, podremos realizar algunas predicciones en cuanto a el cálculo de Y, dado un parámetro X (en este caso las horas de estudio), esto con ayuda de los métodos de nuestro modelo, obteniendo los siguientes resultados, mostrados en la segunda imágen:

```
#we make some predictions

print("Predictions: ")
predict_regr = [[10],[8.5],[6],[4]]

for i in predict_regr:
    print(f"Horas estudiadas {i} puntaje estimado {regr.predict([i])}")
```

```
Predictions:
Horas estudiadas [10] puntaje estimado [[98.63159321]]
Horas estudiadas [8.5] puntaje estimado [[84.05158117]]
Horas estudiadas [6] puntaje estimado [[59.75156109]]
Horas estudiadas [4] puntaje estimado [[40.31154503]]
```

Ya que obtenemos los resultados de predicciones obtenidos con el framework de scikit-learn, procederemos a realizar un análisis más amplio de nuestras variables obtenidas con ayuda de herramientas, y conceptos estadísticos vistos en clase y otros módulos, para entrar más en este contexto, para empezar, se inicia calculando algunos errores en el set de datos 'train' y 'test', obtenidos gracias a la división de la función de split, explicada anteriormente. Para determinar el error, se calcula una resta entre la producción para Y y los datos originales en Y para ambos conjuntos de datos.

```python
#we calculate the prediction error in test and train

y_pred = regr.predict(x_test)
y_pred_train = regr.predict(x_train)
crossValidaton = abs(cross_val_score(regr, x_train, y_train, cv=5).mean())

print("Model score test: ", r2_score(y_test, y_pred))
print("Model score train: ", r2_score(y_train, y_pred_train ))

print("MSE test: ", mean_squared_error(y_test,y_pred))
print("MSE train: ",mean_squared_error(y_train, y_pred_train))
```

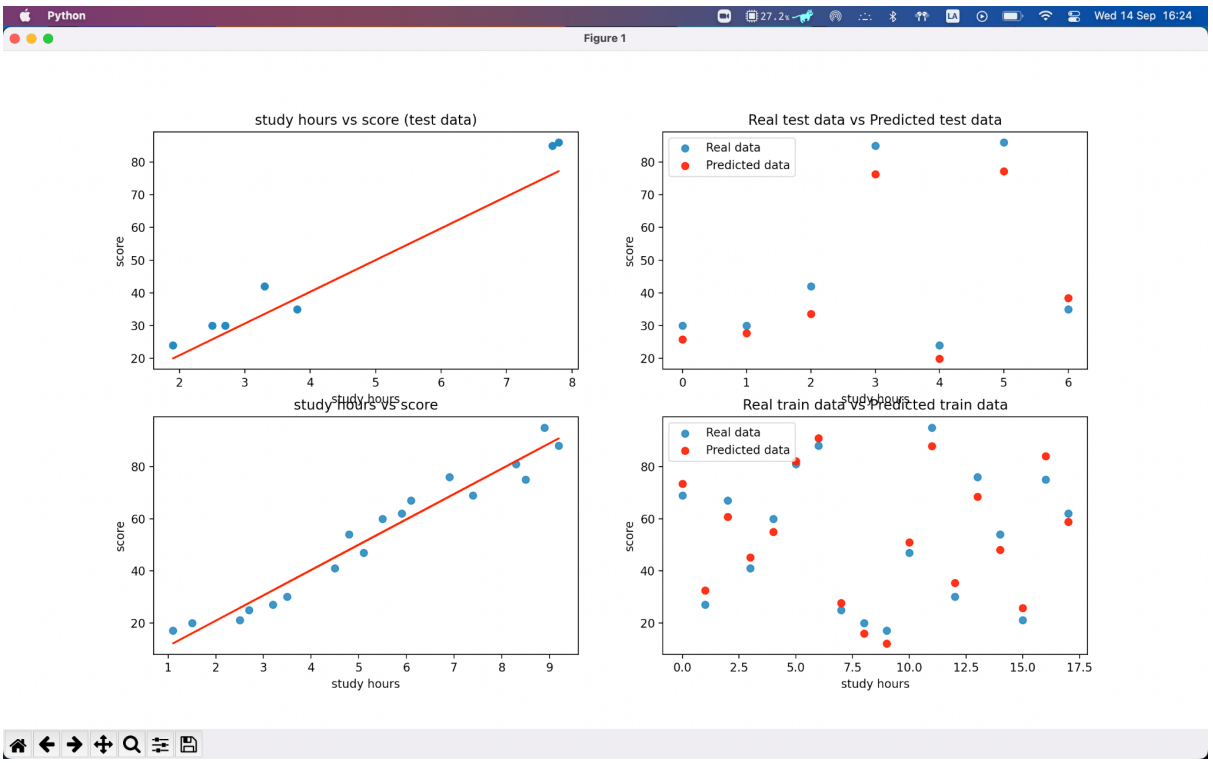
Adicionalmente, para validar que el resultado del entrenamiento, se obtiene, con ayuda del framework el cross validation, teóricamente podemos definir este parámetro como una técnica para evaluar modelos de ML mediante el entrenamiento de varios modelos de ML en subconjuntos de los datos de entrada disponibles y evaluarlos con el subconjunto complementario de los datos, en este caso lo usamos para saber cómo es que se verá afectado el resultado y predicciones de nuestro modelo, dependiendo de la separación de variables en la sección split, dándonos como resultado un valor de 0.93, lo que representa un buen

resultado para poder validar nuestros resultados correspondientes a las predicciones que se realicen con el modelo de regresión lineal.

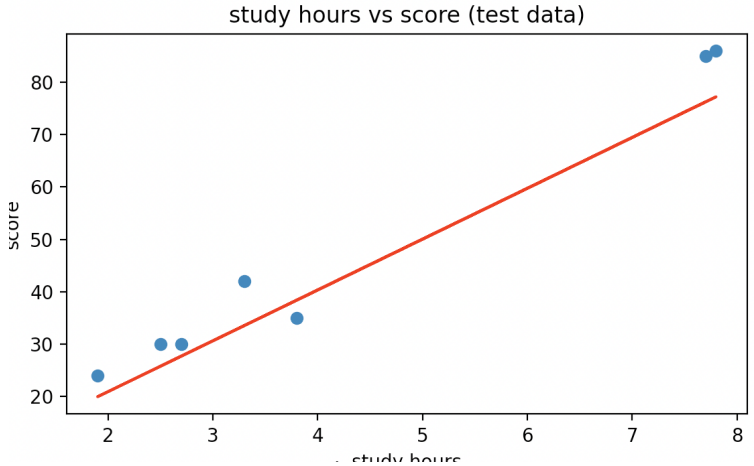
Finalmente, se realiza una graficación de la regresión lineal obtenida, tanto para el set de train y el set de dataset original.

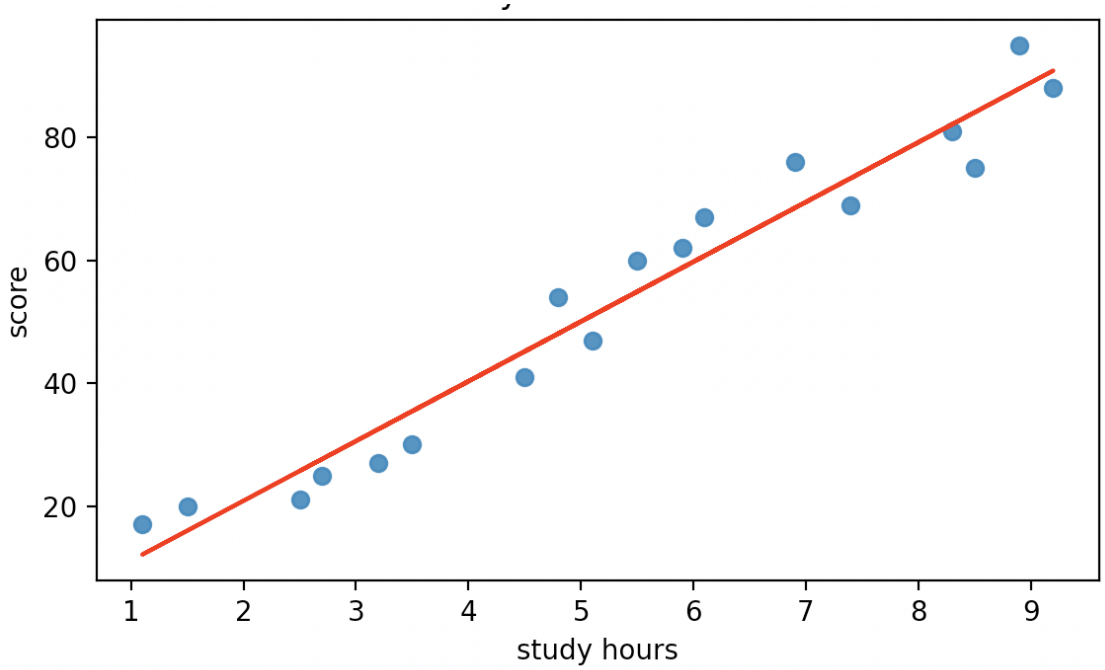
```
#plotting...

figure, axis = plt.subplots(2,2)

#using test

#Lineal regression
axis[0,0].scatter(x_test, y_test)
axis[0,0].plot(x_test, y_pred, color='red')
axis[0,0].set_title("study hours vs score (test data)")
axis[0,0].set(xlabel = 'study hours', ylabel = 'score')


#Var
axis[0,1].scatter(range(len(x_test)), y_test, alpha = 0.9, label = 'Real data')
axis[0,1].scatter(range(len(x_test)), y_pred, color='red',alpha = 0.9, label = 'Predicted data')
axis[0,1].set_title("Real test data vs Predicted test data")
axis[0,1].set(xlabel = 'study hours', ylabel = 'score')
axis[0,1].legend()

#using train

#regresion
axis[1,0].scatter(x_train, y_train, alpha = 0.9)
axis[1,0].plot(x_train, y_pred_train, color='red')
axis[1,0].set_title("study hours vs score")
axis[1,0].set(xlabel = 'study hours', ylabel = 'score')

#var
axis[1,1].scatter(range(len(x_train)), y_train, alpha = 0.9, label = 'Real data')
axis[1,1].scatter(range(len(x_train)), y_pred_train, color='red',alpha = 0.9, label = 'Predicted data')
axis[1,1].set_title("Real train data vs Predicted train data")
axis[1,1].set(xlabel = 'study hours', ylabel = 'score')
axis[1,1].legend()
plt.show()
```

Obtenemos como resultado las siguientes figuras.

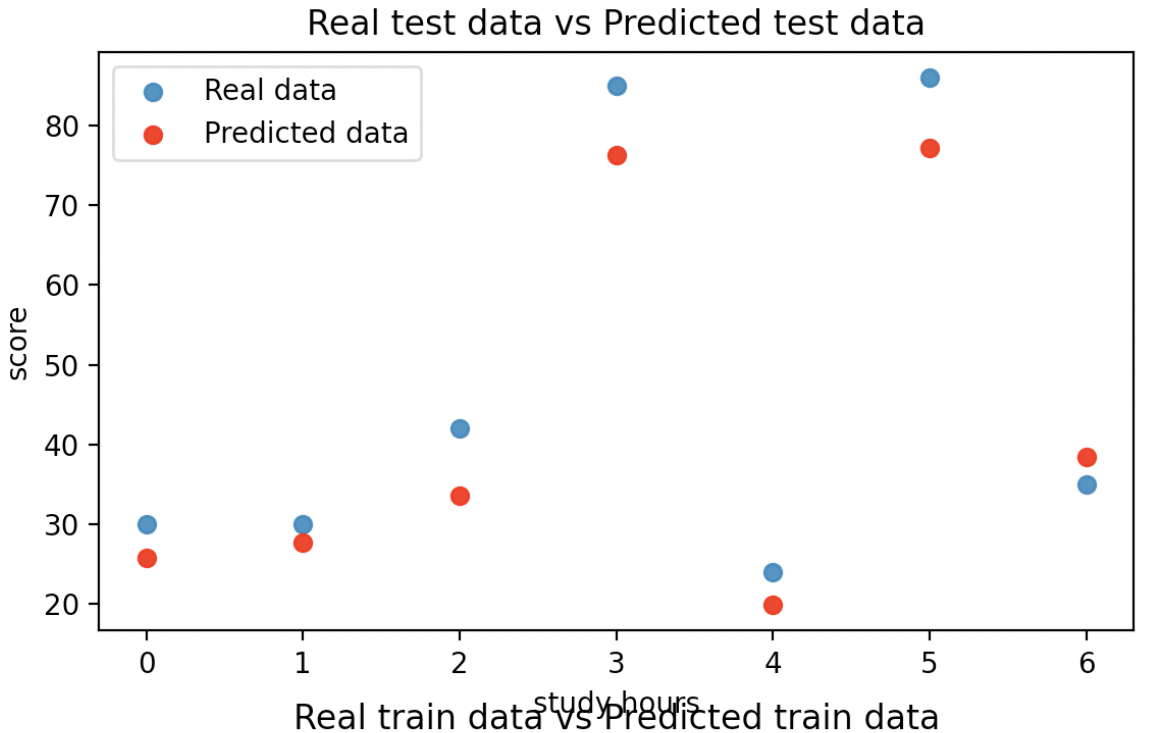


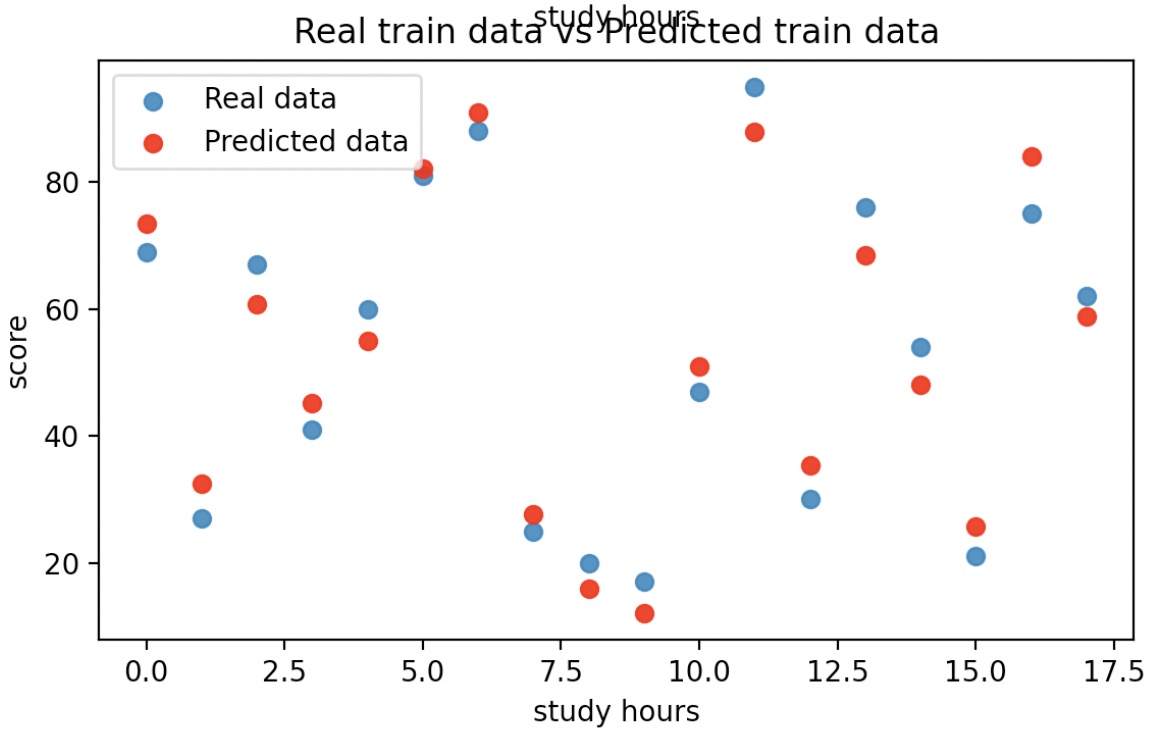
A continuación se procederá a analizar los resultados obtenidos por las gráficas, para empezar, se analizan las primeras dos gráficas, es decir la de el resultado de la regresión lineal, podemos observar que en el test, la línea de pendiente llega a tener una pendiente más elevada, mientras que con el otro conjunto de datos, observamos una pendiente más menos elevada, debido a que se ocupan los valores predecidos.





Ahora bien, analizando un poco las últimas dos gráficas, podemos hacer una comparación entre datos originales, como del train y el test, con respecto a los datos generados con el error en la predicción empleada para cada dataset. La representación de esta manera, nos puede dar una idea aún más clara de la consistencia y el comportamiento de los datos dejando afuera el train y el test. Podemos observar que para cada set, los datos mantienen una tendencia similar en términos de extensión y algunos patrones, confirmando que el modelo tiene un sesgo bajo, pues estos datos (los datos reales y los resultantes del error de predicción) muestran separación entre sí, dándonos indicios de que además, el modelo muestra una varianza alta.


Real test data vs Predicted test data
Real data
Predicted data
score
study hours
20
30
40
50
60
70
80
0
1
2
3
4
5
6


Real train data vs Predicted train data
Real data
Predicted data
score
study hours
20
40
60
80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Para buscar mejoras del modelo empleado, se usará el modelo Ridge, lo que quiere decir que será calculado a través de la función de linear least squares

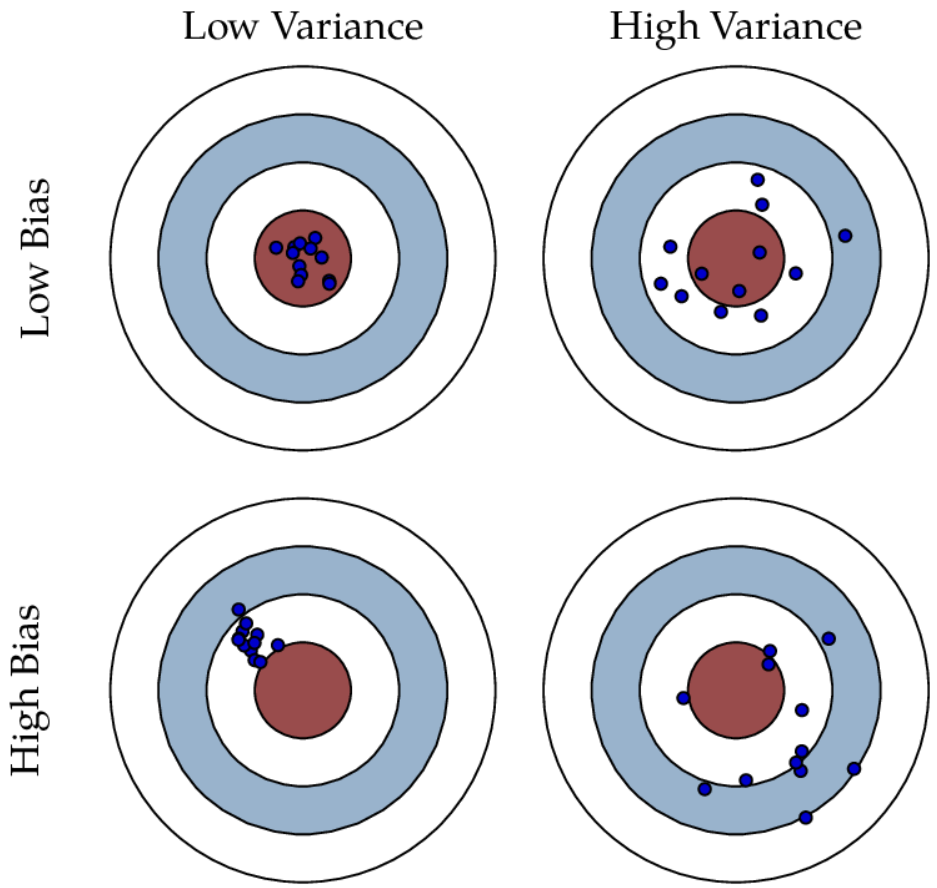
```
#IMPROVING MODEL WITH RIDGE

clf = Ridge(alpha=1.0)
clf.fit(x_train, y_train)

train_ridge_pred = clf.predict(x_train)

print("MSE in ridge train: ", mean_squared_error(y_train, train_ridge_pred))
print("ridge score train: ", r2_score(y_train, train_ridge_pred))

test_ridge_pred = clf.predict(x_test)
print("MSE in ridge test: ", mean_squared_error(y_test, test_ridge_pred))
print("ridge score test: ", r2_score(y_test, test_ridge_pred))
```

En comparación con el modelo de regresión original, los coeficientes MSE y score con ridge mejoran, con lo que podemos decir que de acuerdo a los patrones se repiten en las gráficas de train y del data, a pesar de tomar en cuenta que tiene una alta el set de datos original, según las gráficas vistas arriba.

```
Model score test:  0.9347551352640704
Model score train:  0.9552141872167703
MSE test:  39.51708358434528
MSE train:  27.237100246516963

Cross validaton:  0.927284213912815


MSE in ridge test:  39.53000161965184
MSE in ridge train:  27.28367897683498
ridge model score train:  0.9551375980689968
ridge model score test:  0.9347338068817663
```

Al tomar en cuenta que los resultados obtenidos por este nuevo modelo, podemos ver mejoras, tanto en el mean square error tanto en el conjunto de datos de test y train, y además, tomando en cuenta que el modelo Ridge, se usa generalmente para poder interpretar conjunto de datos (sets de datos) en los cuales se interpretan que tienen una varianza alta, por lo tanto, al ver mejoría con respecto al score original es

decir con Linearregresion(), podemos decir que nuestro modelo tiene una varianza alta.



Podemos decir y concluir que el modelo usado es **fitting**, debido a que tenemos un **low bias** y **high variance** por lo que podemos decir que el modelo es confiable, esto de acuerdo a las medidas vistas en las gráficas, aunado a todas las medidas mostradas, tanto en la mejora del modelo, como en la descripción de la varianza. Por naturaleza, el modelo de regresión lineal corresponde a un modelo de underfitting que se nota más deficiente conforme aumenta la cantidad de datos distribuidos de manera poco óptima para la línea de regresión. Sin embargo, debido a las gráficas mostradas del resultado de la regresión lineal, podemos decir que se comporta de manera correcta, es decir , la pendiente pasa entre los datos obtenidos, de manera constante, es decir, no se ve que haya datos atípicos o extremos.

Áreas de mejora

En las áreas de mejora, definitivamente diría que el campo más importante a mejorar es que haya un dataset más abundante en cuanto a información o datos históricos, ya que al ser de pocos elementos los que tenemos, el modelo podría llegar a ser impreciso, por falta de datos. Y aunque el modelo representa un puntaje bueno arriba de 90%, debemos tomar en cuenta que el cálculo de errores en relación a datasets de mayor complejidad, y la relación de estos nuevos datos y el modelo con la varianza, dejan en claro que puede haber errores al momento de querer aplicarlo en alguna situación de la vida real.

Conclusión

En mi opinión personal, considero que esta ha sido una de las actividades que más he disfrutado realizar en todos estas semanas que llevamos del curso, ya que abarca muchos temas vistos en teoría a programar y analizar a través de gráficas, resultados de cálculos y sobre todo, aprender a usar y buscar librerías que nos van a ayudar a simplificar el trabajo, en cuanto a machine learning concierne.
Considero también que fue un buen ejercicio para poder practicar y aprender un poco más sobre Python en general, así como sus librerías principales y así poder apreciar más estas implementaciones listas para usarse en la vida cotidiana, esto lo pudimos aprender, gracias a la actividad previa, en donde implementamos un algoritmo de regresión lineal, pero sin usar alguna librería de apoyo.

Referencias

https://scikit-learn.org/stable/index.html
https://pandas.pydata.org/docs/index.html
https://matplotlib.org/