

**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Campus Querétaro**

**TC3006C Inteligencia artificial avanzada para la ciencia de datos I**

**Módulo 2 Análisis y Reporte sobre el desempeño del modelo.**

**Profesores:**

Benjamín Valdés Aguirre

Carlos Alberto Dorantes Dosamantes

José Antonio Cantoral Ceballos

Denisse Lizbeth Maldonado Flores

Alejandro Fernández Vilchis

**Presenta:**

José Emiliano Riosmena Castañón – A01704245

**Fecha:**

Sábado, 7 de septiembre del 2024

**Índice**

Tabla de contenido

[**Introducción** 3](#_Toc176608132)

[**Descripción del Dataset** 4](#_Toc176608133)

[**Selección de columnas (features)** 4](#_Toc176608134)

[**Uso de ETL (Extract – Transform – Load)** 6](#_Toc176608135)

[**Implementación sin el uso de framework** 9](#_Toc176608136)

[**Función de escalamiento (Scaling)** 9](#_Toc176608137)

[**Hipótesis** 10](#_Toc176608138)

[**Mean Square Error (MSE)** 11](#_Toc176608139)

[**Gradiente Descendiente** 12](#_Toc176608140)

[**Coeficiente R^2** 13](#_Toc176608141)

[**Resultados (Implementación sin framework)** 15](#_Toc176608142)

[**Resultados MSE** 15](#_Toc176608143)

[**Resultados R^2** 15](#_Toc176608144)

[**Visualización de los resultados (gráficas)** 16](#_Toc176608145)

[**Conclusiones y áreas de oportunidad** 17](#_Toc176608146)

# **Introducción**

En el presente ensayo, se explorará el proceso integral para construir un modelo de regresión lineal, abarcando desde la elección del conjunto de datos adecuado hasta la interpretación detallada de los resultados obtenidos. El análisis incluirá una discusión sobre los usos de las principales técnicas y algoritmos estudiados en clase, los cuales han sido fundamentales para la creación y ajuste del modelo.

Se profundizará en los desafíos encontrados durante la implementación, así como en las soluciones adoptadas para superarlos, resaltando la importancia de realizar ajustes dinámicos en cada etapa. Este proceso de refinamiento continuo asegura no solo un análisis robusto de los datos, sino también su conversión efectiva en información valiosa para la toma de decisiones.

Dibujo con letras blancas

Descripción generada automáticamente con confianza media

# **Descripción del Dataset**

Antes de analizar las técnicas empleadas, es importante familiarizarnos con el dataset utilizado en este proyecto. Comprender el dataset nos ayudará a entender cómo funcionan las técnicas aplicadas a este último.

Para este proyecto, utilicé un dataset obtenido de Kaggle que incluye especificaciones de automóviles de 1985. Entre las características del dataset se encuentran la marca, el tipo de combustible, la tracción de las ruedas, la ubicación del motor, el tipo de vehículo, el cilindraje, el tamaño, entre otras. Este dataset resulta adecuado debido a su variedad de columnas numéricas y categóricas, así como a su amplia cantidad de instancias, lo que facilita su manipulación y permite obtener resultados precisos.

Con esta base, ahora podemos explorar las técnicas utilizadas en el modelo. Empezaremos con ETL, la primera técnica aplicada para extraer, transformar y cargar nuestro dataset.

Tabla

Descripción generada automáticamente con confianza media

**Figura 1.0. Ejemplo de los datos originales del dataset.**

## **Selección de columnas (features)**

Como mencionaba anteriormente, el objetivo del modelo era predecir el precio de un vehículo basado en sus diferentes características. Por lo que el siguiente paso es identificar qué características nos servirían para lograr esta predicción. Para este caso, consideré que las columnas más relevantes para poder hacer esta predicción son las siguientes:

* Distancia entre ejes (*wheel\_base):* Influye en el tamaño y la estabilidad del vehículo, mientras más larga sea esta distancia, los vehículos suelen ser más espaciosos y costosos.
* Peso en vacio (*curb\_weight*): Los vehículos más pesados suelen ser tener precios más altos debido a los costos asociados con materiales y componentes adicionales.
* Tamaño del motor (*engine\_size*): El tamaño de un motor es un indicador directo de la potencia y el rendimiento del vehículo. Generalmente, los motores más grandes suelen ser más caros debido a la capacidad y rendimiento mejorado.
* Potencia (*horsepower*): La potencia del motor nos ayuda a medir el rendimiento del vehículo. Los vehículos con mayor potencia suelen ser más caros ya que ofrecen un rendimiento superior y normalmente están asociados con modelos deportivos o de lujo.
* Millas por galón en ciudad (*city\_mpg*): La eficiencia de combustible en la ciudad puede influir en el precio. Los vehículos que ofrecen mejor rendimiento de combustible en la ciudad pueden tener un precio más alto debido a la demanda por vehículos económicos.
* Millas por galón en carretera (*highway\_mpg*): Similar al caso anterior, la eficiencia del combustible en la carretera puede influir en el precio. Los vehículos que son eficientes en carretera suelen tener un precio mayor debido a su menor costo de operación.

Estas características nos proporcionan una buena representación de aspectos importantes del vehículo, y nos podrían servir para que nuestro modelo pudiera predecir el precio de un vehículo con este conjunto de características.

# **Uso de ETL (Extract – Transform – Load)**

Para el dataset utilizado, lo primero que hicimos fue definir qué información del conjunto de datos nos resultaba útil. En este caso, empleé el dataset para intentar predecir el precio de un vehículo basándome en algunas de sus características. Para ello, usé la librería pandas para extraer los datos del archivo imports-85.data e imports-85.names para identificar las columnas. Luego, combinamos ambos archivos en un dataframe que contenía toda la información, aunque inicialmente de forma desordenada.

Texto

Descripción generada automáticamente

**Figura 2.0. Función para cargar el dataset en un DataFrame**

Con los datos cargados en el dataframe, el siguiente paso fue limpiarlos e identificar qué información sería relevante para construir el modelo. Primero, eliminé los valores vacíos en la columna de precios (price). En este dataset, los valores faltantes estaban representados con un signo de interrogación (?), que pandas interpreta como una cadena de texto (string). Esto podría causar problemas al combinar esos caracteres con los valores numéricos de la columna. Para resolverlo, reemplacé los signos de interrogación por NaN y luego eliminé los valores vacíos.

Después de eso, convertimos las columnas con datos numéricos para evitar conflictos con las cadenas de texto. En nuestro caso, las columnas que presentaban este inconveniente eran la de precios y la de potencia del motor (horsepower).

Luego de seleccionar las columnas que usaríamos como características (features), realizamos la imputación de los valores faltantes en el resto de las columnas. Para ello, calculamos los promedios de cada columna y reemplazamos los valores vacíos por sus respectivos promedios. Con esto, completamos la fase de transformación de los datos.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Figura 2.1. Obtención del promedio de las columnas**

Texto

Descripción generada automáticamente

**Figura 2.2. Sustitución de valores vacíos por los promedios**

En cuanto a la fase de carga, el uso fue limitado. Básicamente, una vez transformados los datos, los empleamos en diferentes algoritmos para las funcionalidades del proyecto, generando análisis e interpretaciones. No obstante, creo que una buena práctica habría sido guardar los datos transformados en un archivo CSV al finalizar el proceso. Esto habría evitado tener que repetir la transformación cada vez que se ejecuta el código, reduciendo la complejidad y el tamaño del archivo final.

Ya hemos hablado sobre los primeros pasos para realizar nuestro análisis. Ahora veremos cómo usamos estos datos para obtener nuestro modelo, qué métodos y algoritmos hemos aplicado para obtener nuestros resultados.

# **Implementación sin el uso de framework**

El primer reto de esta actividad fue implementar uno de los algoritmos vistos en clase sin el uso de un framework como sci-kit learn, etc. En clase, el profesor nos compartió una serie de varios de los algoritmos que veíamos en las clases. Yo decidí utilizar el del gradiente descendiente, pues ese fue el que más llamó mi atención y creí que podría obtener buenos resultados con él. La solución que he construido fue basada de la implementación compartida por nuestro profesor.

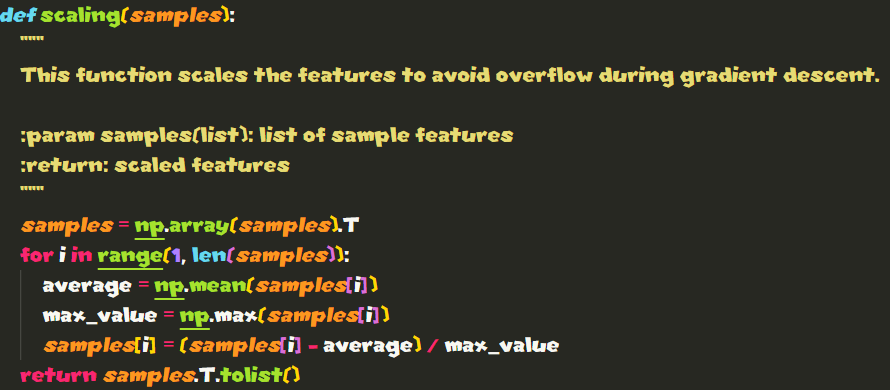
Las principales funciones que destacar sobre esta implementación son las siguientes:

* Escalamiento
* Hipótesis
* MSE
* Gradiente Descendiente

## **Función de escalamiento (Scaling)**

El escalamiento es la transformación de variables predictores para que tengan una escala común. Nos ayuda a mejorar el rendimiento del modelo. Pues en un algoritmo como el del gradiente descendiente, donde las características suelen tener rangos muy diferentes, las actualizaciones de los parámetros podrían ser desiguales y afectaría a la convergencia del modelo. Las técnicas más comunes de escalamiento son:

* Normalización: Ajustamos los valores para que queden dentro de un rango específico.
* Estandarización: Transformamos la variable para que tenga un promedio de 0 y una desviación estándar de 1.

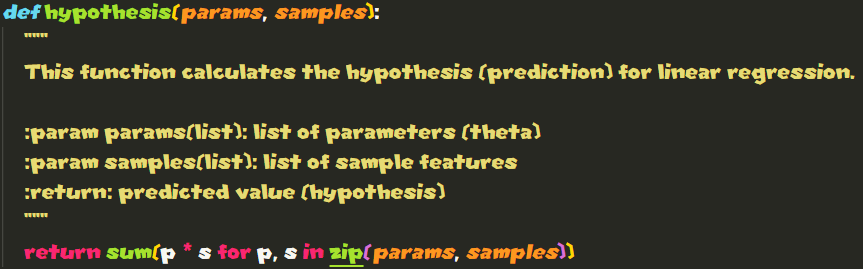


**Figura 3.0. Función de escalamiento**

En la implementación, empleamos una combinación de técnicas que nos permite procesar las características de manera más eficiente. Inicialmente, tomamos la lista de características y la convertimos en arreglos transpuestos, lo que nos facilita trabajar con cada una de ellas de forma individual. A continuación, calculamos tanto el promedio como el valor máximo de cada característica. Estos valores nos sirven de base para normalizar los datos. Realizamos nuestro escalamiento, a cada muestra le restamos el promedio y lo dividimos entre el valor máximo. De esta manera, ajustamos los datos para que estén dentro de un rango más uniforme, facilitando el análisis posterior y mejorando el rendimiento de los modelos predictivos que dependen de estos datos.

## **Hipótesis**

Esta se refiere a la suposición sobre la relación entre las variables. En regresión línea, esta establece que la variable dependiente es una función lineal de las variables independientes. Esta se expresa de la siguiente forma:



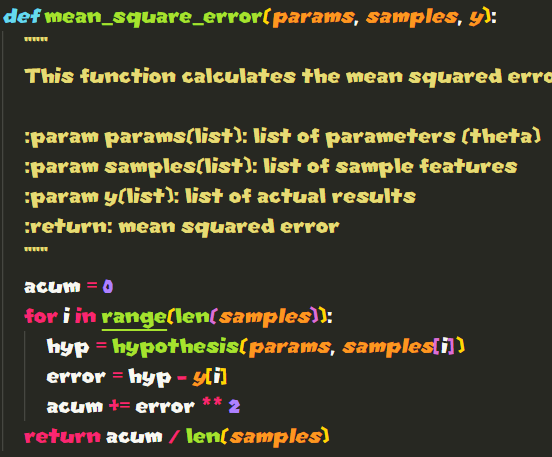
**Figura 3.1. Función para calcular la hipótesis**

En la implementación, la función es bastante sencilla. Lo que hacemos es multiplicar cada elemento en params por su característica en simples. Y finalmente hacemos la sumatoria de estos productos para obtener nuestra hipótesis.

## **Mean Square Error (MSE)**

Es una métrica que se usa para medir el rendimiento de un modelo de regresión. Evalúa qué tan bien se ajustan las predicciones del modelo a valores reales, calculando el promedio de los cuadrados de los errores. Su fórmula es la siguiente:

Cuando tenemos errores muy grandes, estos tienen un fuerte impacto en el MSE, ya que los errores se elevan al cuadrado y los convierte en potenciales outliers. Cuando tenemos un MSE bajo, nos dice que las predicciones del modelo están cercanas a los valores reales. Por otro lado, un MSE alto nos dice que las predicciones no son precisas.



**Figura 3.2. Función MSE**

En nuestra función, utilizamos la función de la hipótesis para calcular nuestros errores. Al resultado de la hipótesis le restamos su valor real de Y, y con eso obtenemos el valor de nuestro error, estos los elevamos al cuadrado y los vamos acumulando para finalmente dividirlos entre el número de muestras de nuestro modelo.

## **Gradiente Descendiente**

El gradiente descendiente es un algoritmo de optimización utilizado para minimizar funciones de costo en problemas de aprendizaje automático en la regresión. Su objetivo es encontrar los parámetros del modelo que minimizan el error entre las predicciones del modelo y los valores reales.

La fórmula para actualizar los parámetros del modelo es el siguiente:

El gradiente es el vector de derivadas parciales de la función de costo con respecto a cada parámetro, se mide cómo cambia la función de costo si se modifica un parámetro específico. Los parámetros del modelo se actualizan en la dirección opuesta al gradiente para reducir la función de costo. Repetimos este proceso por un número definido de iteraciones o que la función de costo converja a un valor lo suficientemente bajo.

Texto

Descripción generada automáticamente

**Figura 3.3. Algoritmo del gradiente descendiente**

En la implementación, utilizamos una variable temporal para almacenar los nuevos valores después de realizar una actualización, de ese modo evitamos modificar la lista original durante la iteración. Recorremos cada parámetro y calculamos el gradiente usando la suma de los errores multiplicados por el valor de la característica correspondiente. Los parámetros en temp se actualizan después de restar el producto del gradiente y la tasa de aprendizaje, y finalmente regresamos esos parámetros actualizados.

## **Coeficiente R^2**

El coeficiente de determinación es una medida estadística que indica qué tan bien se ajusta el modelo a los datos. Nos muestra la proporción de la variabilidad de la variable dependiente que puede ser explicada por las variables independientes del modelo. Su fórmula es:

Un valor de R^2 de alto nos dice que el modelo puede explicar la mayor parte de la variabilidad de los datos, lo que indica un buen ajuste. Por otro lado, un valor de R^2 bajo nos dice que el modelo no puede explicar con precisión la mayor parte de la variabilidad. Lo que nos puede indicar que nuestro modelo no es útil, o incluso decirnos que es peor usar el modelo que usar la media como predicción.

Texto

Descripción generada automáticamente

**Figura 3.4. Función para determinar el R^2 del modelo**

La implementación de nuestra función resume lo que hemos explicado anteriormente. Calculamos el coeficiente de determinación al realizar la suma de los valores reales menos el promedio, elevados al cuadrado, y la suma de los valores reales menos los valores predichos, elevados al cuadrado, y hacemos la resta de uno menos el cociente de la suma total de errores cuadráticos sobre la suma total de los cuadrados.

# **Resultados (Implementación sin framework)**

A lo largo de este ensayo, hemos hablado sobre el dataset utilizado, que características tiene, y cómo lo usaríamos. También hemos hablado de cómo hemos construido nuestras funciones para tratar de predecir el precio de un automóvil basado en las características mencionadas anteriormente.

Ahora vamos a ver los resultados que hemos obtenido en esta primera ejecución, veremos cuales son los valores de R^2 para train y validation; el valor de nuestro MSE y una visualización de la gráfica de train y validation. Hemos establecido un total de Epochs de 1000 con una tasa de aprendizaje de 0.9.

## **Resultados MSE**

Imagen que contiene Código QR

Descripción generada automáticamente

**Figura 4.0. Resultados de MSE**

En los últimos epochs del modelo, observamos que el error en Train disminuye, lo que indica que el modelo está aprendiendo mejor los patrones en los datos de entrenamiento. Sin embargo, también notamos que el error en Validation aumenta, lo cual sugiere que el modelo está perdiendo su capacidad de generalizar a datos nuevos. Esto ocurre porque, en lugar de continuar aprendiendo los patrones subyacentes, el modelo comienza a memorizar o ajustarse al ruido del conjunto de entrenamiento, lo que reduce su rendimiento en los datos de validación. Este comportamiento indica la presencia de **overfitting**.

## **Resultados R^2**

Pantalla de video juego

Descripción generada automáticamente con confianza media

**Figura 4.1. Resultados de los coeficientes R^2**

Estos son los resultados de nuestros coeficientes R^2 en Train y Validation. La diferencia entre estos dos nos revela puntos importantes sobre el desempeño del modelo:

* Train → 0.7568: Significa que el modelo puede explicar aproximadamente el 75.68% de la variabilidad en los datos de entrenamiento. Es un ajuste relativamente bueno, pues sugiere que el modelo esta capturando los patrones presentes en esos datos.
* Validation → 0.4635: Significa que el modelo solo explica el 46.35% de la variabilidad en los datos nuevos. Este valor es más bajo que el de entrenamiento, nos dice que el modelo no está generalizando bien a datos nuevos.

Esta diferencia nos dice que, aunque el modelo se ajusta relativamente bien a los datos que le hemos dado, pierde precisión cuando se enfrenta a datos nuevos, indicando una capacidad limitada de generalización. Como mencionábamos anteriormente, el modelo está ajustándose a detalles específicos de los datos de entrenamiento, pero estos detalles no son representativos de los datos de validación. Por ende, el rendimiento en validación es peor de lo que se esperaba.

## **Visualización de los resultados (gráficas)**

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Figura 4.2. Gráficas de los errores en train y validation**

La gráfica muestra el comportamiento del error en Train y en Validation a lo largo de los epochs del modelo.

Por un lado, en Train el error de entrenamiento es bastante alto al principio, sugiriendo que el modelo tiene un bajo rendimiento en los primeros pasos del entrenamiento. Sin embargo, conforme el entrenamiento avanza, el error disminuye de forma rápida y se estabiliza alrededor de los 300 epochs. Como hemos mencionado, esta reducción sugiere que el modelo mejora su capacidad para ajustarse a los datos de entrenamiento.

Por otro lado, en Validation al principio, este también disminuye, lo que indica que el modelo mejoraba su rendimiento en datos no vistos. Sin embargo, poco antes de los 100 epochs, el modelo deja de disminuir y aumenta en algunos puntos. Como mencionamos, nuestra modelo está sobre ajustado a los datos de entrenamiento. Por lo que, aunque el modelo mejore en su rendimiento con los datos de entrenamiento, pierde su capacidad para generalizar datos nuevos.

## **Conclusiones y áreas de oportunidad**

En esta primera implementación, lo qué podemos concluir, es lo que hemos mencionado a lo largo de estos resultados. A medida que los epochs avanzan, el modelo está aprendiendo demasiado bien los datos de entrenamiento, incluyendo los patrones específicos y el ruido, pero pierde su capacidad para generalizar a nuevos datos. Lo que nos dice que nuestro modelo tiene overfitting. Unas posibles formas de mejorar el desempeño del modelo es aplicar técnicas de regularización como Ridge o Lasso para evitar que el modelo se ajuste demasiado a los datos de entrenamiento. Otra solución podría ser detener el entrenamiento cuando el error de validación comience a aumentar, antes de que el modelo se sobreajuste.

# **Implementación con el uso de framework**

Para esta segunda implementación. Hemos vuelto a aplicar el algoritmo del gradiente descendiente pero ahora usando un framework de desarrollo, Sci-kit learn. Esta librería nos permite usar la función del gradiente descendiente con el uso de una sola línea en lugar de generar toda una función completa, además de las otras funciones usadas en la implementación pasada.

Como en la implementación sin framework, hemos vuelto a aplicar la técnica de ETL para la extracción de datos, siendo esa la única función que tenemos presente en nuestro código y donde se realiza lo mismo que hicimos anteriormente.

A continuación, veremos las funciones destacadas de nuestra implementación.