Práctica 4. Regresión Lineal y Logística

Universidad Nacional Autónoma de México
Facultad de Ingeniería
Reconocimiento de Patrones (757-1)
Basile Álvarez Andrés José
Keller Ascencio Rodolfo Andrés
Shen Shuai
*

***Resumen*—A lo largo de este documento se presentarán los resultados y análisis de nuestra cuarta práctica de la materia de Reconocimiento de Patrones, donde se estará trabajando con la implementación de la función de costo y los algoritmos de descenso de gradiente para problemas que se resuelvan con regresión lineal. De misma manera, se trabajará con una aplicación de regresión logística y se responderán preguntas relacionadas con la importancia de una buena elección de tasa de aprendizaje, las implicaciones en la elección de los parámetros iniciales theta, así como el proceso y las herramientas que nos permiten encontrar un mínimo global absoluto para el conjunto de datos trabajado.**

I. INTRODUCCIÓN

I-A. Regresión Lineal

La regresión lineal es una técnica de análisis de datos que predice el valor de datos desconocidos mediante el uso de otro valor de datos relacionados y conocido. En este sentido, se modela matemáticamente la variable dependiente y la variable independiente como una ecuación lineal.[1] En otras palabras, es una técnica de modelado estadístico que se emplea para describir una variable de respuesta continua como una función de una o varias variables predictoras.[2]

Como se mencionó anteriormente, las técnicas de regresión lineal permiten crear un modelo lineal que describe la relación entre una variable dependiente como una función de una o varias variables independientes Xi. La ecuación general correspondiente a un modelo de regresión lineal es: $Y = \beta_0 + \sum \beta_i X_i + \epsilon_i$, donde β representa las estimaciones de parámetros lineales que se deben calcular y ϵ_i representa los términos de error.[2]

Existen diversos tipos de regresión:

- Regresión lineal simple.
- Regresión lineal múltiple.
- Regresión lineal multivariante.
- Regresión lineal múltiple multivariante

A lo largo de esta práctica se buscará reconocer los principios y fundamentos de la regresión lineal múltiple al trabajar con dos variables predictoras "Xz una respuesta z.ª predecir.

I-B. Regresión Logística

Por otro lado, otra técnica de análisis de datos que existe actualmente es la regresión logística, siendo un modelo estadístico utilizado para estudiar las relaciones entre un conjunto de variables predictoras "Xz una respuesta z.ª predecir, al igual que la regresión lineal. La diferencia entre ambos tipos de regresiones radica en el hecho de que en la regresión logística, normalmente, la predicción tiene un número finito de resultados, como un sí o un no, siendo generalmente utilizado para la clasificación de dos conjuntos de datos.

Para que esto sea posible, la regresión logística hace uso de la función sigmoide:

$$y = \frac{1}{1 + e^{-x}}$$

Esta función sigmoide nos permite predecir la probabilidad de que un evento ocurra o no, a partir de la optimización de los coeficientes de regresión. Los resultados obtenidos tras la aplicación de este modelo se encuentran siempre dentro de un rango que va de 0 a 1. Para poder proceder a realizar la clasificación tras la aplicación del algoritmo, se hace uso de un umbral, a partir del cual en caso de que los valores obtenidos pasen por encima del valor del umbral, se clasifican dentro de una categoría, y en caso de encontrarse por debajo de este umbral se encontrarán dentro de otra categoría. Entre más alejados se encuentren los valores dle umbral, más precisa será la clasificación del modelo, siendo esto lo que nos interesaría obtener al momento de entrenar nuestro modelo de regresión logística.[3]

I-C. Tasa de aprendizaje

Dentro del modelado de datos, la tasa de aprendizaje se refiere al valor que afecta a la velocidad en la que el algoritmo alcanza o converge en las ponderaciones óptimas.[4] Para el caso de los modelos de regresión lineal y logística que se verán a lo largo de la práctica, la tasa de aprendizaje se refiere al número por el cual se multiplicarán los parámetros de la recta para realizar pequeñas aproximaciones de la recta al punto.[1]

Para el caso de la elección de nuestra tasa de aprendizaje, la teoría nos indica que al momento de escoger tasas de aprendizaje muy grandes, los pasos o cambios en la optimización de parámetros serán también muy grandes y sería posible pasarnos de largo con respecto a los valores óptimos buscados, haciendo que al momento de volver a plantear la dirección óptima que buscamos para la actualización de parámetros tardemos en llegar debido a los cambios tan grandes que se realizan. Por el contrario, si decidimos definir una tasa de

aprendizaje muy pequeña, los cambios serán mínimos siendo más difícil y computacionalmente costoso el aproximarnos al valor mínimo buscado, por lo cual realizaríamos muchas más operaciones para llegar a él a comparación de los casos en los cuales definamos una tasa de aprendizaje intermedia.

A lo largo de la práctica se profundizará acerca de estas ideas.

I-D. Conjunto de datos utilizado

Para la realización de esta práctica, se trabajó con dos conjuntos de datos distintos, uno para la regresión lineal y otro para el desarrollo de la aplicación de regresión logística.

Para el caso del ejercicio de regresión lineal se trabajó con el conjunto:

$$x = [[1,0],[1,0{,}5],[1,1],[1,1{,}5],[1,2],[1,2{,}5],[1,3],[1,4],[1,5]]$$

$$y = [0,0{,}5,1,1{,}5,2,2{,}5,3,4,5]$$

siendo este un conjunto de datos utilizados para que en el desarrollo de la práctica fuera más sencillo comprender el funcionamiento de la regresión lineal. Por otro lado, para el caso de la implementación de la regresión logística se hizo uso del conjunto de datos *diabetes.csv*, el cual contaba con 768 registros que clasificaban a pacientes con diabetes a partir de ocho variables, entre las cuales se evaluaba información relacionada con la edad del paciente, presión, nivel de insulina, entre otros datos. A partir de estas variables, se obtenían dos posibles tipos de clasificaciónes, pacientes con diabetes y pacientes sin diabetes.

preg	plas	pres	skin	insu	mass	pedi	age	class
6	148	72	35	0	33.6	0.627	50	tested_positive
1	85	66	29	0	26.6	0.351	31	tested_negative
8	183	64	0	0	23.3	0.672	32	tested_positive
1	89	66	23	94	28.1	0.167	21	tested_negative
0	137	40	35	168	43.1	2.288	33	tested_positive
5	116	74	0	0	25.6	0.201	30	tested_negative
3	78	50	32	88	31	0.248	26	tested_positive
10	115	0	0	0	35.3	0.134	29	tested_negative
2	197	70	45	543	30.5	0.158	53	tested_positive
8	125	96	0	0	0	0.232	54	tested_positive
4	110	92	0	0	37.6	0.191	30	tested_negative
10	168	74	0	0	38	0.537	34	tested_positive
10	139	80	0	0	27.1	1.441	57	tested_negative
1	189	60	23	846	30.1	0.398	59	tested_positive
5	166	72	19	175	25.8	0.587	51	tested_positive
7	100	0	0	0	30	0.484	32	tested_positive
0	118	84	47	230	45.8	0.551	31	tested_positive
7	107	74	0	0	29.6	0.254	31	tested_positive
1	103	30	38	83	43.3	0.183	33	tested_negative
1	115	70	30	96	34.6	0.529	32	tested_positive
3	126	88	41	235	39.3	0.704	27	tested_negative
8	99	84	0	0	35.4	0.388	50	tested_negative
7	196	90	0	0	39.8	0.451	41	tested_positive
9	119	80	35	0	29	0.263	29	tested_positive
11	143	94	33	146	36.6	0.254	51	tested_positive
10	125	70	26	115	31.1	0.205	41	tested_positive
7	147	76	0	0	39.4	0.257	43	tested_positive
1	97	66	15	140	23.2	0.487	22	tested_negative
13	145	82	19	110	22.2	0.245	57	tested_negative
5	117	92	0	0	34.1	0.337	38	tested_negative
5	109	75	26	0	36	0.546	60	tested_negative
3	158	76	36	245	31.6	0.851	28	tested_positive
3	88	58	11	54	24.8	0.267	22	tested_negative

Figura 1: Algunos de los registros tomados del conjunto de datos *diabetes.csv*.

I-E. Herramientas utilizadas

Para el desarrollo de esta práctica, específicamente se pedía que la implementación de la solución al problema se realizara en el lenguaje de programación Python, dado que se proporcionaba un código de inicio para obtener una base desde la cual comenzar a realizar la resolución, donde se esperaba que dentro de los fragmentos de código se actualizara los que se encontraran marcados por una simbología .ªQUÍ USTED...". Los específicos de la implementación utilizando el lenguaje Python serán vistos a lo largo del desarrollo y resultados de la práctica, así como en el apartado IV de este documento.

II. OBJETIVO

El objetivo principal de esta práctica se enfoca en que el alumno logre comprender e implementar la función de costo y los algoritmos de descenso de gradiente para la regresión lineal y logística.

III. DESARROLLO

III-A. ¿Qué sucede si la tasa de aprendizaje es demasiado baja?

La tasa de aprendizaje en el algoritmo de descenso por gradiente es un parámetro que controla la velocidad a la que se ajustan los parámetros θ iniciales a los valores de θ óptimas. En cada iteración se ajustan los parámetros en la dirección opuesta al gradiente multiplicando por la tasa de aprendizaje. Un valor bajo de α hará que los ajustes sean más pequeños, lo que puede llevar a una convergencia lenta del algoritmo, o que no llegue a los parámetros θ óptimas antes del número máximo de pasos.

Por ejemplo, si cambiamos la tasa de aprendizaje a 0.0005, obtuvimos los siguientes valores: $\theta_0 = 1{,}8027$ y $\theta_1 = 0{,}4162$.

Gráficamente, observamos que los resultados anteriores no son los valores óptimos, ya que no se ajustan a los datos iniciales.

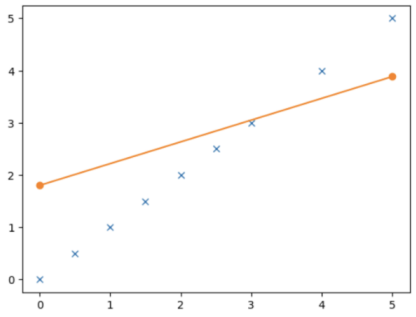


Figura 2: Tasa de aprendizaje baja.

Sin embargo, si aumentamos el número máximo de iteraciones, en este caso maxsteps = 100000, observamos que los resultados vuelven a ajustarse a los datos iniciales.

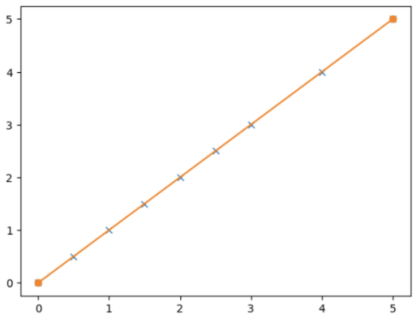


Figura 3: Tasa de aprendizaje baja, número de iteraciones alto.

III-B. ¿Qué sucede si la tasa de aprendizaje es demasiado alta?

Una tasa de aprendizaje demasiado alta puede ocasionar un ajuste rápido de los parámetros, pero también puede provocar la divergencia del algoritmo en lugar de su convergencia. En otras palabras, en lugar de acercarse a los valores óptimos, el algoritmo puede oscilar alrededor del mínimo global, generar inestabilidad numérica como desbordamiento o números NAN (Not a number), y aumentar de forma descontrolada. En el mejor de los casos, puede alcanzar una convergencia subóptima, llegando a un mínimo local pero no al mínimo global deseado.

Por ejemplo, si utilizamos una tasa de aprendizaje demasiado alta, por ejemplo, 5 o 0.5, podemos experimentar problemas de inestabilidad numérica.

```
print("Optimized Theta0 is ", thet[0])
print("Optimized Theta1 is ", thet[1])

Optimized Theta0 is  nan
Optimized Theta1 is  nan
```

Figura 4: Tasa de aprendizaje baja, número de iteraciones alto.

III-C. ¿Puede la regresión lineal realmente encontrar el mínimo global absoluto?

No es posible encontrar el mínimo global absoluto en muchos de los casos o problemas que intentamos resolver en la vida real, ya que los valores no siempre se pueden representar con una función lineal o hay una estructura compleja en los datos que no se puede capturar por un modelo lineal.

Probablemente, el punto inicial desde el cual comencemos nuestro análisis de regresión lineal para una función no convexa lleve a un punto mínimo local (que no representa el punto óptimo de solución del problema). No obstante, en estos casos se pueden escoger múltiples puntos iniciales y después comparar los puntos mínimos encontrados con el descenso de gradiente, quedándonos con el punto que tenga el menor valor.

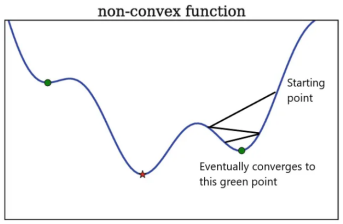


Figura 5: Encontrar el mínimo global absoluto implicaría tener otro punto de inicio para nuestro problema de regresión lineal.

Otro problema que puede evitar que lleguemos a un mínimo global es la selección de la tasa de aprendizaje, ya que, de ser muy grande, podemos "saltar.[el] mínimo global y no converger, y de ser muy pequeño podríamos tardar demasiado tiempo en llegar a él.

No obstante, al utilizar la regresión lineal con función de costo de error mínimo cuadrático (MSE), el problema de optimización se ve como una función convexa, lo cual indica que únicamente hay un mínimo (el mínimo global). Por lo tanto, si el algoritmo de optimización de descenso de gradiente converge a un punto mínimo, podemos asegurar que es el mínimo global.

Podemos ver, matemáticamente, que la función de costo de MSE describe una parábola (función convexa) y que al optimizar con descenso de gradiente sobre esta función, llegaremos a un único punto mínimo (mínimo global).

$$\textit{Least Squared Error} = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

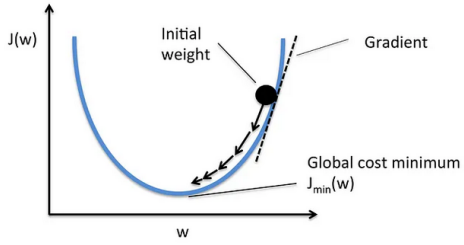
$$\textit{Cost Function} = J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \quad m = \textit{number of sample data}$$

Figura 6: Encontrar el mínimo global absoluto implicaría tener otro punto de inicio para nuestro problema de regresión lineal.

derivando, para hallar el punto mínimo global:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

lo cual puede ser visto gráficamente en la siguiente figura:



III-D. ¿Qué efecto tiene si cambia la suposición inicial de thet0 y thet1 para el descenso del gradiente a algo completamente fuera de lugar?

III-D1. Parámetros Theta0=2 y Theta1=0:

- Optimized Theta0 is 3.985292837903086e-07
- Optimized Theta1 is 0.9999998725702243

III-D2. Parámetros Theta0=0 y Theta1=1:

- Optimized Theta0 is 0.0
- Optimized Theta1 is 1.0

III-D3. Parámetros Theta0=0 y Theta1=0:

- Optimized Theta0 is 5.493254475113613e-08
- Optimized Theta1 is 0.9999999824353137

III-D4. Parámetros Theta0=2000 y Theta1=1000:

- Optimized Theta0 is 0.0002887191268974
- Optimized Theta1 is 0.9999076820321958

III-D5. Parámetros Theta0=200000 y Theta1=-100000:

- Optimized Theta0 is 0.039852983310057086
- Optimized Theta1 is 0.9872570048626341

Actualización de theta					
Theta0	Theta1	Th0 Opt	Th1 Opt	%Err Th0	%Err Th1
2	0	0.000000398	0.99999987	0.000039	0.001274
0	1	0.0	1.0	0.0	0.0
0	0	0.000000054	0.99999998	0.000005	0.000176
2000	1000	0.0002887	0.99990768	0.028871	0.923265
200000	-100000	-0.039852873	0.98725700	3.985287	1.290748
-200000	100000	0.0000003	1.01274296	0.000039	1.258262

Cuadro I: Actualización con valores iniciales fuera de lugar

III-D6. Parámetros Theta0=-200000 y Theta1=100000:

- Optimized Theta0 is -0.03985287344496788
- Optimized Theta1 is 1.0127429600079934

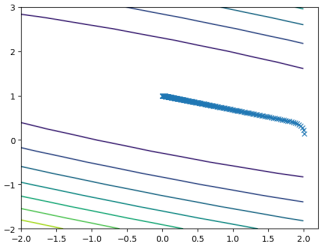


Figura 7: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de 2 y theta 1 de 0.

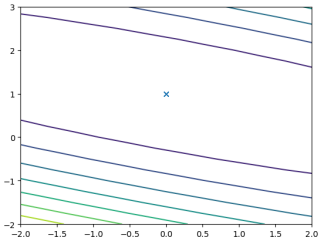


Figura 8: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de 0 y theta 1 de 1.

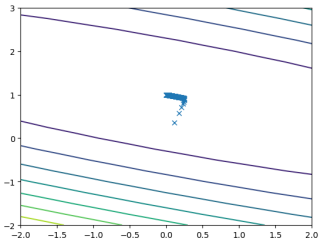


Figura 9: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de 0 y theta 1 de 0.

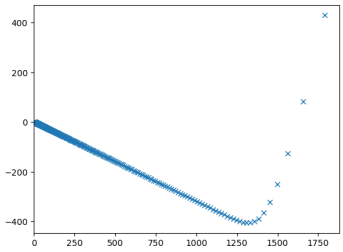


Figura 10: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de 2000 y theta 1 de 1000.

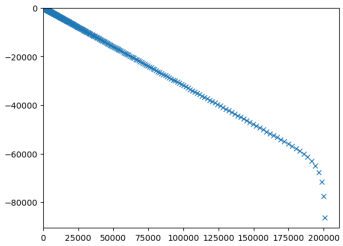


Figura 11: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de 200000 y theta 1 de -100000.

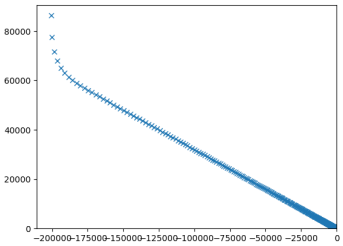


Figura 12: Grafica de las soluciones encontradas por el algoritmo de Descenso de Gradiente para apreciar la aproximación al mínimo deseado al iniciar con un valor theta0 de -200000 y theta 1 de 100000.

A partir de lo visto de la experimentación, y gracias a lo leído en diversas referencias podemos concluir varios aspectos de este ejercicio, por ejemplo, en caso de que inicialicemos las variables theta0 y theta1 en un mínimo local o global, sus valores se preservarán y no harán cambios o actualizaciones, lo cual pudimos apreciar al momento en el que les dimos el valor theta0 = 0 y theta1 = 1. [5]

Otra propiedad que podemos evaluar a partir de los experimentos realizados es el hecho de que a pesar de que inicialicemos ambos datos de theta con el mismo valor, al iterar estos valores van a verse modificados y no se mantendrán iguales, buscando la optimización para cada uno.[5]

Por otro lado, otro de los experimentos realizados sería en caso de que inicializáramos theta con valores muy grandes

y alejados. En caso de los datos trabajados en la práctica pudimos apreciar que el plano de la función de costo y, en sí, la función obtenida a partir de los datos utilizados es relativamente poco compleja, por lo cual no se tuvo tanto problema más que el aumento en el error de los valores de theta óptimo obtenidos, sin embargo, en caso de trabajar con un conjunto de datos más complejo que modele una función con varios valles y crestas, los valores de theta óptimos podrían verse atrapados en uno de estos valles y no salir del mínimo local, en lugar de obtener el valor del mínimo global de la función, por lo que es importante una correcta aproximación de los valores de theta al inicializarlos.[5]

En conclusión, para el conjunto de datos del ejercicio a pesar de que ponemos valores de theta fuera de lugar, siendo estos muy grandes o chicos, estos valores se van actualizando hasta llegar a un valor óptimo. Para el caso de los valores dados por el notebook original, al cambiar los valores iniciales de theta el resultado sería que para theta0 nos acercamos a un valor próximo a 0 y para theta 1 a un valor próximo a 1, esto sucede debido a la naturaleza del conjunto de datos que es muy simple para el análisis del proceso de regresión lineal. Logramos llegar a los valores óptimos de theta0 = 0 y theta1 = 1 únicamente en el caso en el que inicializamos theta con estos valores, esto se debe a que en todos los demás casos al estar trabajando con métodos numéricos de aproximación, nos vamos acercando a estos valores pero no llegamos al óptimo en ningún momento.

III-E. ¿Qué sucede si no se actualiza thet0 y thet1 "simultáneamenteçomo debería, sino que se actualizan ambos parámetros en lazos separados?

Si los valores de theta y theta1 se actualizan de manera independiente, puede llevar a resultados incorrectos o subóptimos. El objetivo de la regresión lineal es estimar los coeficientes de theta y theta1 que minimicen el error entre la predicción y el valor real, generalmente utilizando el descenso de gradiente. El descenso de gradiente actualiza los coeficientes de manera simultánea tomando pasos proporcionales a los gradientes negativos de la función de pérdida con respecto a cada coeficiente. Esta actualización simultánea asegura que los coeficientes se muevan en la dirección que minimiza colectivamente la función de pérdida y converja hacia la solución óptima.

Si modificamos el código para actualizar los valores de theta y theta1 de forma independiente, llegamos a los siguientes resultados:

III-F. ¿Cuántas iteraciones del algoritmo de descenso de gradiente tiene que realizar para alcanzar los valores exactos correctos de theta0 y theta1?

El número de iteraciones que se deben realizar para alcanzar los valores exactos correctos de theta0 y theta1 de forma práctica es muy variable dependiendo de la tasa de aprendizaje utilizada y de los valores iniciales de theta0 y theta1. En sí, no encontramos los valores correctos exactos, debido a que al estar trabajando con métodos numéricos logramos

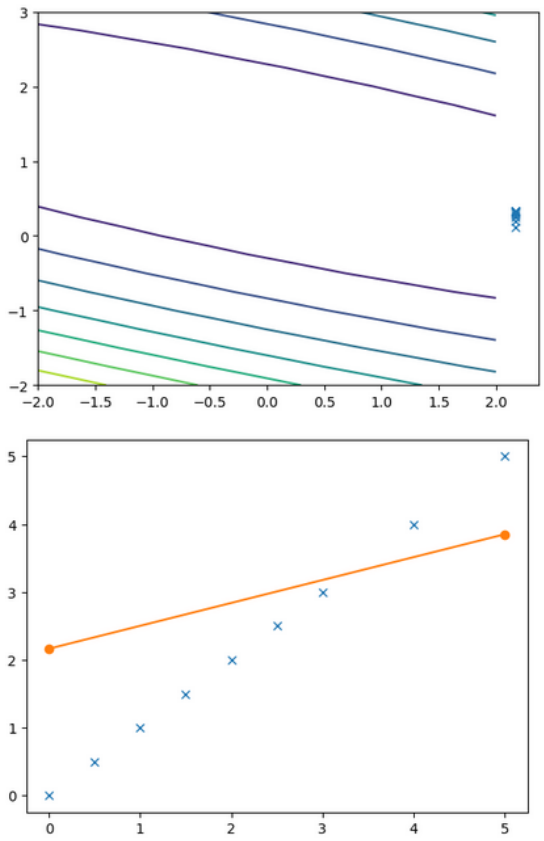


Figura 13: Al actualizar los coeficientes de theta0 y theta1 de manera independiente, vemos que el descenso de gradiente no llega al mínimo (no encontramos la solución óptima).

aproximarnos bastante, sin embargo, se mantiene cierto rango de error al momento de querer obtener estos valores.

Para este proceso, decidimos probar indistintamente con valores altos para el número de iteraciones a trabajar. Tras trabajar con mil, cinco mil, diez mil, cien mil y un millón de iteraciones de prueba nos dimos cuenta que a partir de las diez mil iteraciones con una tasa de aprendizaje de 0.05 y valores iniciales para theta0 y theta 1 de 2 y 0, respectivamente, ya no obteníamos cambios en la salida del valor final optimizado de theta0 y theta1.

A partir de la definición del máximo número de iteraciones a utilizar para probar empíricamente este ejercicio, creamos una función que nos permitiera evaluar los valores históricos de theta por cada iteración, los cuales se almacenan dentro de la variable thetaHist. Tras hacer uso de la función *np.allclose*, se compararon las filas entre sí y mediante un valor de tolerancia encontramos qué tanto cambiaban entre sí dichos resultados.

Con una tolerancia de 1e-23 encontramos que el valor de las thetas ya no cambiaba a partir de la iteración 3017. Dependiendo de la toleracia que utilizábamos este valor se veía modificado, sin embargo, al momento de trabajar con esta tolerancia y tolerancias mayores a 1e-23 ya no encontramos cambios a partir de una mayor tolerancia.

Al acceder a cada uno de los elementos dentro del historial de las thetas, encontramos que a partir de la iteración 5621 ya no encontramos cambios, sin embargo, esto se realizó accediendo manualmente a cada una de las iteraciones obtenidas y validando el momento en el cual los valores de theta ya no se veían modificados con respecto al valor de theta óptimo alcanzado tras diez mil iteraciones. Se tuvo que realizar de forma manual debido a que al ser un error o diferencia tan pequeño ya no se calculaba correctamente con otras funciones utilizadas en python.

En la sección IV se detallará la función utilizada para el cálculo de convergencia.

III-G. Regresión logística - Caso de diabetes

Realizamos las siguientes modificaciones para cambiar el algoritmo de regresión lineal a regresión logística:

- En la función `gradientDescent`, actualizamos la forma en que calculamos las predicciones. En lugar de calcular las predicciones directamente como `predictions = np.dot(x, theta)`, ahora aplicamos la función sigmoide a la salida de la función lineal: `predictions = sigmoid(np.dot(x, theta))`.
- En la función `costFunction`, utilizamos la función sigmoide para calcular las predicciones como `predictions = sigmoid(z)`, donde z es el resultado de la multiplicación de X por θ.
- En la función `costFunction`, calculamos el costo utilizando la fórmula de la regresión logística, que implica el cálculo de la función logarítmica y la función sigmoide.

Una vez realizadas las modificaciones necesarias, importamos los datos crudos de los pacientes con diabetes. Para poder trabajar con la columna de la clase, la transformamos en valores numéricos, asignando 1 a los casos positivos y 0 a los negativos. Posteriomente, dividimos los datos en conjuntos de entrenamiento y prueba, entrenamos nuestro modelo de regresión logística utilizando una tasa de aprendizaje de 0.01, un número máximo de iteraciones de 1000 y θ iniciales igual a [0, 0, 0, 0, 0, 0, 0, 0]. Los θ óptimos obtenidos fueron [1.17802637, 0.31741276, -1.05008194, -0.06815892, 0.00170839, 0.22688641, 0.05184131, 0.37874229]. Luego, utilizando los datos de prueba, calculamos el costo que resultó en 4.0271 y obtuvimos una exactitud de 70 %.

Además, obtuvimos el modelo de regresión logística utilizando la función LogisticRegression de la biblioteca sklearn, donde los θ óptimos obtenidos fueron [0.06454373, 0.03409222, -0.01387313, 0.00328315, -0.00180345, 0.10261653, 0.62730352, 0.03706176] y logramos una precisión del 74 %, .

IV. CÓDIGO

Durante el desarrollo de esta práctica, se utilizó Google Colab como herramienta de gestión de versiones del código, así como método para trabajar de forma remota con el equipo. Se presentan algunos elementos importantes del código a continuación:

IV-A. Importando las bibliotecas que vamos a utilizar

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

IV-B. Definición de función para el descenso de gradiente

```
# Diapositiva 22, Gradient Descent algorithm
def gradientDescent(x, y, theta, alpha, m, maxsteps): # x valores independientes, y valores dependientes,
↪  theta parametro de aprendizaje, Alpha tasa de aprendizaje, m ejemplos de entrenamiento, max numero
↪  maximo de iteraciones
    thetaHist = np.empty([maxsteps, 2])
    xTrans = x.transpose()

    for i in range(0, maxsteps):
        predictions = np.dot(x, theta)        # h(xi) Obtener predicciones con los valores de theta inicial
        error = predictions - y               # h(xi)-yi Obtener el error entre la predicción y valores
        ↪  reales (Objetivos)
        gradient = np.dot(xTrans, error) / m  # XT(h(xi)-yi) Obtener el gradiente
        theta = theta - alpha * gradient      # -(/j) = -(XT(h(xi)-yi)) Obtener nuevos valores de thetas
        thetaHist[i] = theta

    return theta, thetaHist
```

IV-C. Definición de función de costo

```
#Diapositiva 22, Linear Regression Model - Cost Function
def costFunction(x, y, theta):
    predictions = np.dot(x, theta)                    # h(xi)
    error = predictions - y                           # h(xi)-yi
    cost = np.sum(np.square(error)) / (2 * len(y))    # (1/2m)(h(xi)-yi)
    loss = np.sqrt(cost)
    return cost, loss
```

IV-D. Definición de datos de entrenamiento para la regresión lineal

```
x=np.array([[1, 0], [1, 0.5], [1, 1], [1, 1.5], [1, 2], [1, 2.5], [1, 3], [1, 4], [1, 5]])
y=np.array([0, 0.5, 1, 1.5, 2, 2.5, 3, 4, 5])
```

IV-E. Cálculo del tamaño del conjunto de entrenamiento

```
m, n = np.shape(x)
```

IV-F. Graficando el conjunto de datos de entrenamiento

```
fig = plt.figure(1)  # An empty figure with no axes
plt.plot(x[:,1], y, 'x') # Segundo valor de X vs Y
```

IV-G. Calculando los valores para la función de costo

```
theta0 = np.arange(-2, 2.01, 0.25)
theta1 = np.arange(-2, 3.01, 0.25)

#Inicializar la matriz J
J = np.zeros((len(theta0), len(theta1)))

# Calculate values of the cost function
for i in range(0, len(theta0)):
    for j in range(0, len(theta1)):
        cost, loss = costFunction(x, y, [theta0[i], theta1[j]]) #Usar costFunction con diferentes valores
        ↪  de Theta
        J[i, j] = cost #Almacenar el costo en una matriz

print(np.shape(J))
print(J[0][0])
```

IV-H. Graficando la función de costo para su visualización

```
theta0, theta1 = np.meshgrid(theta0, theta1)
fig2 = plt.figure(2)
ax = fig2.add_subplot(121, projection="3d")
surf = ax.plot_surface(theta0, theta1, np.transpose(J))
ax.set_xlabel('theta 0')
ax.set_ylabel('theta 1')
ax.set_zlabel('Cost J')
ax.set_title('Cost function Surface plot')

ax = fig2.add_subplot(122)
contour = ax.contour(theta0, theta1, np.transpose(J))
ax.set_xlabel('theta 0')
ax.set_ylabel('theta 1')
ax.set_title('Cost function Contour plot')

fig2.subplots_adjust(bottom=0.1, right=1.5, top=0.9)
```

IV-I. Implementando la función de descenso de gradiente

```
alpha = 0.05        # learning parameter
maxsteps= 10000     # number of iterations that the algorithm is running

# First estimates for our parameters
thet = [2, 0]

thet, thetaHist, i = gradientDescent(x, y, thet, alpha, m, maxsteps)
print(i)
```

IV-J. Implementando la función de descenso de gradiente con actualización de Theta0 y Theta1 independiente

```
  def gradientDescentIndependentUpdate(x, y, theta, alpha, m, maxsteps):  # x valores independientes, y
  ↪  valores dependientes, theta parametro de aprendizaje, Alpha tasa de aprendizaje, m ejemplos de
  ↪  entrenamiento, max numero maximo de iteraciones
    thetaHist = np.empty([maxsteps, 2])
    xTrans = x.transpose()

    for i in range(0, maxsteps):
        predictions = np.dot(x, theta)        # h(xi) Obtener predicciones con los valores de theta inicial
        error = predictions - y               # h(xi)-yi Obtener el error entre la predicción y valores
        ↪  reales (Objetivos)

        gradient0 = np.dot(xTrans[0], error) / m  # XT(h(xi)-yi) Obtener el gradiente para theta0
        theta[0] = theta[0] - alpha * gradient0   # -(/0) = -(XT(h(xi)-yi)) Actualizar theta0

        thetaHist[i] = theta

    for i in range(0, maxsteps):
        predictions = np.dot(x, theta)        # h(xi) Obtener predicciones con los valores de theta inicial
        error = predictions - y               # h(xi)-yi Obtener el error entre la predicción y valores
        ↪  reales (Objetivos)

        gradient1 = np.dot(xTrans[1], error) / m  # XT(h(xi)-yi) Obtener el gradiente para theta1
        theta[1] = theta[1] - alpha * gradient1   # -(/1) = -(XT(h(xi)-yi)) Actualizar theta1

        thetaHist[i] = theta

    return theta, thetaHist
```

IV-K. Definición e implementación de una función para evaluar convergencia de los valores de theta (Ejercicio 6)

```
def checkConvergence(thetaHist):
    for i in range(1, thetaHist.shape[0]):
        if np.allclose(thetaHist[i], thetaHist[i-1], atol=1.0e-23):
            break
    return i
i = checkConvergence(thetaHist)
print("Convergencia alcanzada en la iteración:", i)
```

IV-L. Impresión de valores óptimos de theta

```
print("Optimized Theta is ", thetaHist[5621])

#Comparando con valores finales
print("Optimized Theta0 is ", thet[0])
print("Optimized Theta1 is ", thet[1])
```

IV-M. Gráfica y visualización de la solución obtenida al problema

```
fig3 = plt.figure(3)
plt.contour(theta0, theta1, np.transpose(J))
plt.plot(thetaHist[:,0], thetaHist[:,1], 'x')
ax.set_xlabel('theta 0')
ax.set_ylabel('theta 1')

# Finally, let's plot the hypothesis function into our data
xs = np.array([x[0,1], x[x.shape[0]-1,1]])
h = np.array([[thet[1] * xs[0] + thet[0]], [thet[1] * xs[1] + thet[0]]])
plt.figure(1)
plt.plot(x[:,1], y, 'x')  # Data
plt.plot(xs, h, '-o')     # hypothesis function
plt.show()
```

IV-N. Funciones de regresión logística

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def gradientDescent(x, y, theta, alpha, m, maxsteps):
    thetaHist = np.empty([maxsteps, len(theta)])
    xTrans = x.transpose()

    for i in range(0, maxsteps):
        z = np.dot(x, theta)
        h = sigmoid(z)
        error = h - y
        gradient = np.dot(xTrans, error) / m
        theta -= alpha * gradient
        thetaHist[i] = theta

    return theta, thetaHist
def costFunction(X, Y, theta):
    m = len(Y)
    z = np.dot(X, theta)
    predictions = sigmoid(z)
    epsilon = 1e-10  # Pequeña corrección para evitar divisiones por cero
    cost = (-1/m) * (np.dot(Y, np.log(predictions + epsilon)) + np.dot((1-Y), np.log(1-predictions +
    ↪  epsilon)))
    return cost
```

IV-Ñ. División de datos

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, :-1]  # Características (todas las columnas excepto la última)
y = df.iloc[:, -1]   # Etiquetas (última columna)

# División en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

IV-O. Implementando regresión logística con datos de entrenamiento

```
alpha = 0.01
maxsteps= 1000
theta_initial = [0,0,0,0,0,0,0,0]

thet, thetaHist = gradientDescent(X, Y, theta_initial, alpha, len(Y), maxsteps)
print(thet)
```

IV-P. Métricas

```
cost = costFunction(X_t, Y_t, thet)
print("Costo:", cost)

predictions = sigmoid(np.dot(X_t, thet))
predictions = np.where(predictions >= 0.5, 1, 0)
accuracy = accuracy_score(Y_t, predictions)
print("Exactitud:", accuracy)
```

IV-Q. Implementando regresión logística con datos de entrenamiento y bibliotecas de python

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
theta = model.coef_
accuracy = accuracy_score(y_test, predictions)
print(theta)
print("Exactitud:", accuracy)
```

V. CONCLUSIONES

A lo largo de esta práctica, se tuvo como principal objetivo el que el alumno lograra comprender e implementar la función de costo y los algoritmos de descenso de gradiente para la regresión lineal y logística a partir de un código de inicio en python. De esta forma, se buscaba que los alumnos reconocieran la importancia y la construcción de una función de costo y una función de descenso de gradiente, así como la manera en cómo se implementaban ambas funciones en problemas reales, tanto en aplicaciones de problemas de regresión lineal y logística.

Por otro lado, la práctica también buscaba que los alumnos fueran capaces de reconocer la importancia de la elección correcta de valores de tasa de aprendizaje y de inicialización de parámetros theta para su optimización para la evaluación correcta de la función a partir de los datos trabajados.

Por parte de la correcta elección de valores de tasa de aprendizaje, se comprendió el que no es correcto tomar valores de tasas de aprendizaje muy grandes o pequeños, puesto que a tasas muy grandes los cambios en los parámetros son muy grandes, lo que llevaría a que al momento de buscar la optimización de los coeficientes de la función que representa al conjunto de datos, estos coeficientes cambien de forma muy abrupta, con lo cual no se llegaría a un valor óptimo debido a los grandes saltos que daría provocando una divergencia, por otro lado, al seleccionar valores de tasa de aprendizaje muy pequeños, los cambios en el valor de los coeficientes serían mínimos, y se tardaría mucho tiempo en llegar a valores óptimos, lo cual además de traducirse en un gasto de recursos de cómputo, se ve traducido en un mayor tiempo de ejecución para dicha optimización, o inclusive en la finalización del cálculo antes de haber llegado a valores óptimos.

Por otro lado, para el caso de la correcta elección de valores de inicialización de theta, al momento de seleccionar los valores óptimos no se tendría ningún cambio al pasar estos valores a través de la función de descenso de gradiente; si escogemos valores muy grandes o fuera del rango del cual se encuentran los óptimos, tardará en aproximarse a los valores óptimos, se tendrán errores grandes o inclusive estos parámetros podrían verse atrapados en mínimos locales en caso de contar con funciones más complejas. Por todo esto es importante un correcto análisis del conjunto de datos que nos encontramos trabajando y, en sí, de la función generada a partir de dicho conjunto de datos.

Con respecto a la actualización simultánea de los parámetros theta, esto resulta necesario debido a que en caso contrario se pueden obtener valores erróneos, gracias a que al momento de realizarse esta actualización se ven afectados ambos parámetros, y en caso de una actualización independiente, estos cambios van a realizarse de forma aislada, con lo cual el algoritmo subre de retrasos y errores.

Finalmente, tras trabajar con el proceso para la realización de una regresión lineal, el código se modificó para trabajar con regresión logística. Para esto, decidimos realizar el proceso tanto de forma manual así como aplicando una función propia de python. Tras haber realizado estas modificaciones y haber trabajado con el conjunto *diabetes.csv*, obtuvimos una precisión del 70 % con nuestra propia implementación y una precisión del 74 % con la implementación de la biblioteca de python, con lo cual podemos considerar que realizamos una correcta implementación, pese a que también creemos que este modelo no es el óptimo para la evaluación y clasificación de este tipo de datos, siendo que se pueden obtener modelos más precisos haciendo uso de árboles de clasificación, bosques aleatorios de clasificación o inclusive redes neuronales, aunque el objetivo de la práctica se centraba en el aprendizaje de la implementación de los modelos de regresión lineal y logística así como en la importancia de un correcto manejo de las tasas de aprendizaje y de los valores de inicialización.

En conclusión, consideramos que esta práctica fue excelente debido a que nos permitió comprender con mayor detalle la importancia de una correcta implementación de los modelos de regresión lineal y logística, así como nos permitió jugar con parámetros para poder reconocer la importancia de una buena elección de valores y una correcta implementación del modelo.

REFERENCIAS

[1] A. W. S. AWS, "¿qué es la regresión lineal?. https://aws.amazon.com/es/what-is/linear-regression/," 2023.

[2] MathWorks, "¿qué es la regresión lineal?. https://la.mathworks.com/discovery/linear-regression.html," 2023.

[3] DataScientest, "¿qué es la regresión logística?. https://datascientest.com/es/que-es-la-regresion-logistica," 2023.

[4] V. Roman, "Machine learning supervisado: Fundamentos de la regresión lineal. https://medium.com/datos-y-ciencia/machine-learning-supervisado-fundamentos-de-la-regresi %c3 %b3n-lineal-bbcb07fe7fd: :text=esta %20tasa %20de %20aprendizaje %20es, de %20la %20recta %20al %20punto." 2023.

[5] mGalarnyk, "Machine learning week 1 quiz 2 (linear regression with one variable) stanford coursera. https://gist.github.com/mgalarnyk/cc964bea99b09e3c733b339ad3b7b019," 2023.