UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE
SAN FRANCISCO XAVIER DE CHUQUISACA

LABORATORIO N°: 4

Aplicación de la Regularización

NOMBRE: Sandra Villca Señoranis

CARRERA: Ing. en Ciencias de la Computación

MATERIA: Inteligencia Artificial I

1. Aplicación de la Regularización a la Regresión Lineal Múltiple

Se utiliza la regresion lineal multiple para predecir el precio de barcos de motor y de velo. El dataset contiene las siguientes características (features):

Columna1 - X1 : Año de fabricación del barco
Columna2 - X2: Largo del barco en pies
Columna3 - X3: Ancho del barco en pies
Columna4 - X4: Cantidad de motores del barco
Columna5 - X5: Potencia del barco en HP
Columna6 - **y**: Precio del Barco en Dólares

1.1 Se ha agregado el término de Regularización a la función de Costo y a la función de Descenso por el Gradiente:

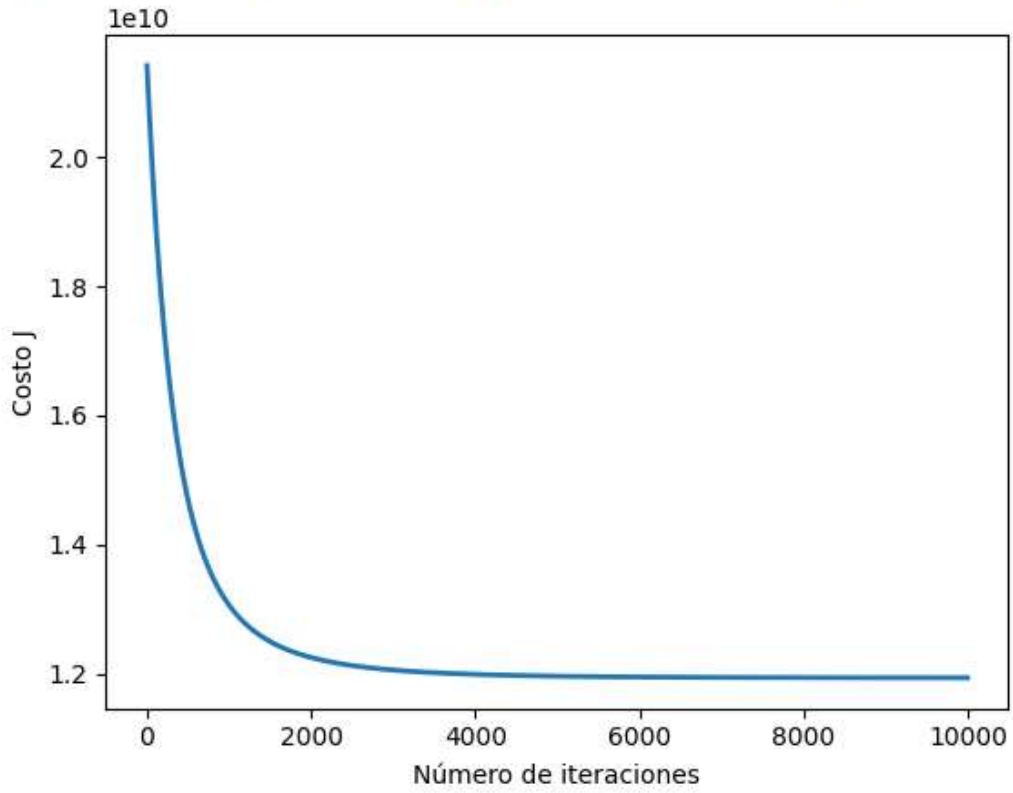
```python
def computeCostMulti(X, y, theta, lambda_reg):
    m = y.shape[0]  # numero de ejemplos de entrenamiento

    h = np.dot(X, theta)

    # Calcular el término de costo sin regularización
    J = (1/(2 * m)) * np.sum(np.square(h - y))

    # Agregar el término de regularización
    reg_term = (lambda_reg / (2 * m)) * np.sum(np.square(theta[1:]))  # Excluye theta_0
    J += reg_term

    return J
```

```python
def gradientDescentMulti(X, y, theta, alpha, lambda_reg, num_iters):
    m = y.shape[0]  # numero de ejemplos de entrenamiento

    # realiza una copia de theta, el cual será actualizado por el descenso por el gradiente
    theta = theta.copy()

    temp = theta
    temp[0] = 0

    J_history = []

    for i in range(num_iters):
        # Actualiza theta sin regularización
        theta = theta - (alpha / m) * ((np.dot(X, theta) - y).dot(X)) - (alpha * lambda_reg / m) * temp

       J_history.append(computeCostMulti(X, y, theta, lambda_reg))

   return theta, J_history
```

1.2 Observaciones del uso de la Regularización

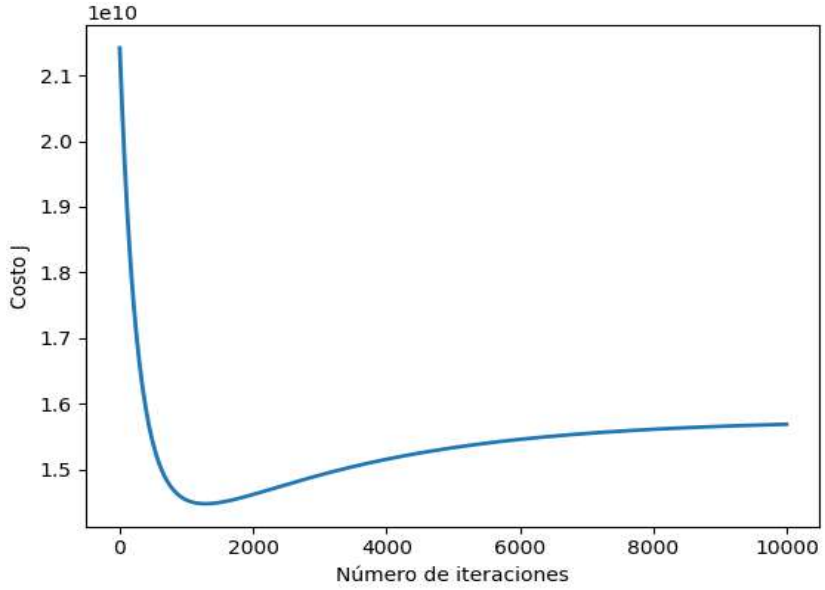
Para valores de:
lambda_reg = 0.1
alpha = 0.001
num_iters =10000
Los thetas calculados, la predicción del precio de barco y la curva de costo no varían en relación a las funciones sin regularización.

```
Theta calculado por el descenso por el gradiente: [ 64589.9509701   23336.66592533  33886.5448(
    243.66147486 108467.03121042]
El precio predecido del barco (usando el descenso por el gradiente con regularización): $20589
```



Haciendo lambda_reg = 10000:

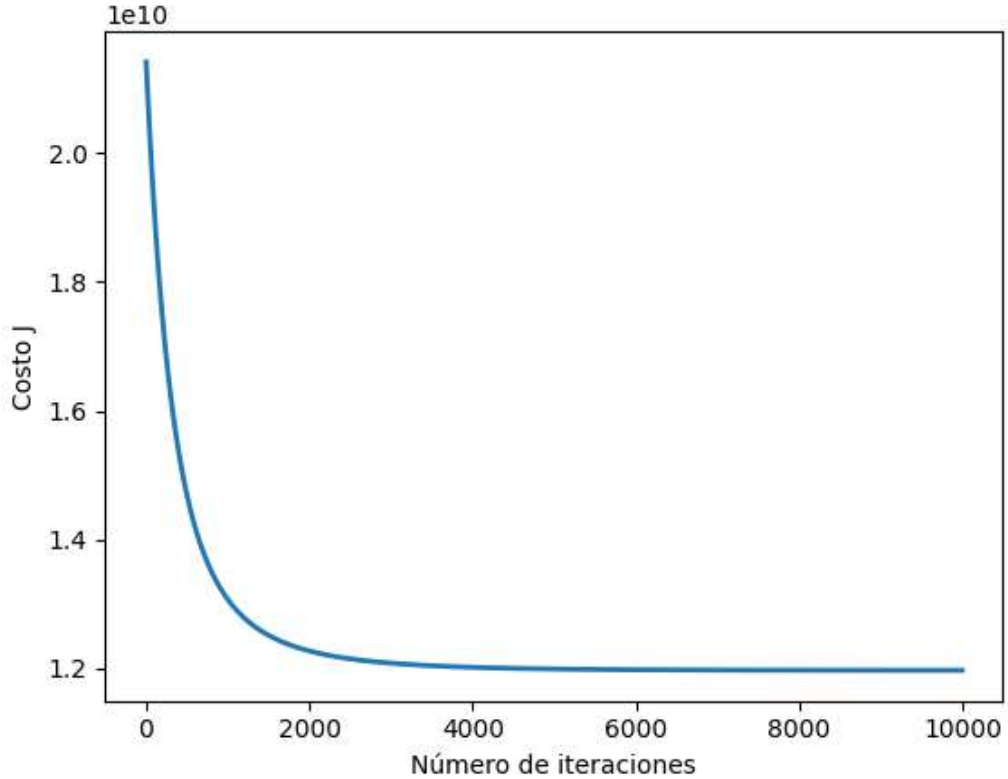
La curva de costo varía considerablemente y a partir de 1000 iteraciones el error crece.



Valores de Lambda_reg entre 0.1 y 100:

Preciden un costo similar al de las funciones sin regularización. La ‘y’ predicha no varía mucho, se mantiene sus valores.

```
Theta calculado por el descenso por el gradiente: [ 64589.9509701   23336.66592533  33886.54480
    243.66147486 108467.03121042]
El precio predecido del barco (usando el descenso por el gradiente con regularización): $20589
```



1.3 Aplicación de la ecuación de la Normal con la Regularización

```
def normalEqn(X, y, lambda_reg):
    # Construye la matriz de regularización L
    L = np.eye(X.shape[1])
    L[0, 0] = 0  # No regularizar el parámetro de intercepción theta0

    # Calcula theta usando la ecuación normal regularizada
    theta = np.dot(np.linalg.inv(np.dot(X.T, X) + lambda_reg * L), np.dot(X.T, y))

    return theta
```

Las predicciones de 'y' (Costo del barco) mejoran pero no significativamente.

Se ha encontrado que el valor óptimo para lambda_reg = 5000.

Con éste valor de lambda_reg, se tiene ésta predicción:

```
X_array = [1, 1992, 21, 8.5, 1, 150]

price = np.dot(X_array, theta)

print('Precio predecido del Barco (usando la ecuación de la normal): $
{:.0f}'.format(price))
```

```
Theta calculado a partir de la ecuación de la normal: [-4.58283506e+06
2.25595080e+03 2.41287792e+03 3.46203224e+02 -6.48725252e+02 3.81940763e+02]
Precio predecido del Barco (usando la ecuación de la normal): $21274
```

Precio predecido sin regularización: `$21733`

Valor verdadero: $16500.

2. Aplicación de la Regularización a la Regresión Polinomial

```
Para la regresión polinomail se aplicó una hipótesis de segundo grado.
```

```
Para las 5 Características, la hipóstesis en función de θ y X es:
hθ(X1,X2,……,X5) = θ0 +  θ1X1 +  θ2X2 +  θ3X3 +  θ4X4 +  θ5X5 +
θ6(X1^2) +  θ7(X2^2) +  θ8(X3^2) +  θ9(X4^2) +  θ10(X5^2) +
θ11(X1X2) + θ12(X1X3) + θ13(X1X4) + θ14(X1X5) +
θ15(X2X3) + θ16(X2X4) + θ17(X2X5) +
θ18(X3X4) + θ19(X3X5) +
θ20(X4X5)
```

La regresión Polinomial se adecúa y se procesa como una regresión lineal múltiple con las mismas funciones de costo y descenso por el gradiente de **1.**

2.1 Observaciones de utilizar la Regularización

```
# Elegir un valor para lambda_reg (parámetro de regularización)
```

```
lambda_reg = 0.1
# Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.001                          #0.001
num_iters = 10000
```

El valor óptimo para lambda_reg = 0.1.

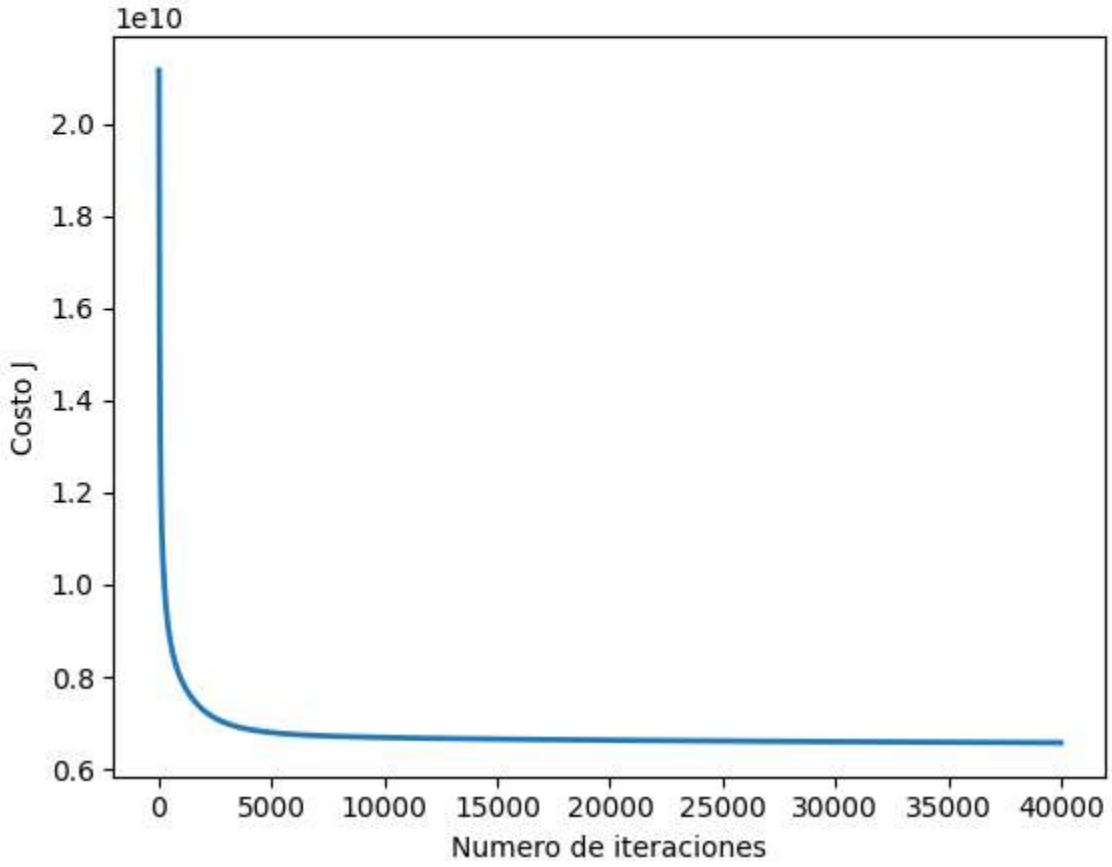
Agregando la regularización se logra reducir el costo en menos iteraciones,

10000 iteraciones con Regularización y 40000 iteraciones sin regularización.

Ver gráfico siguiente página.

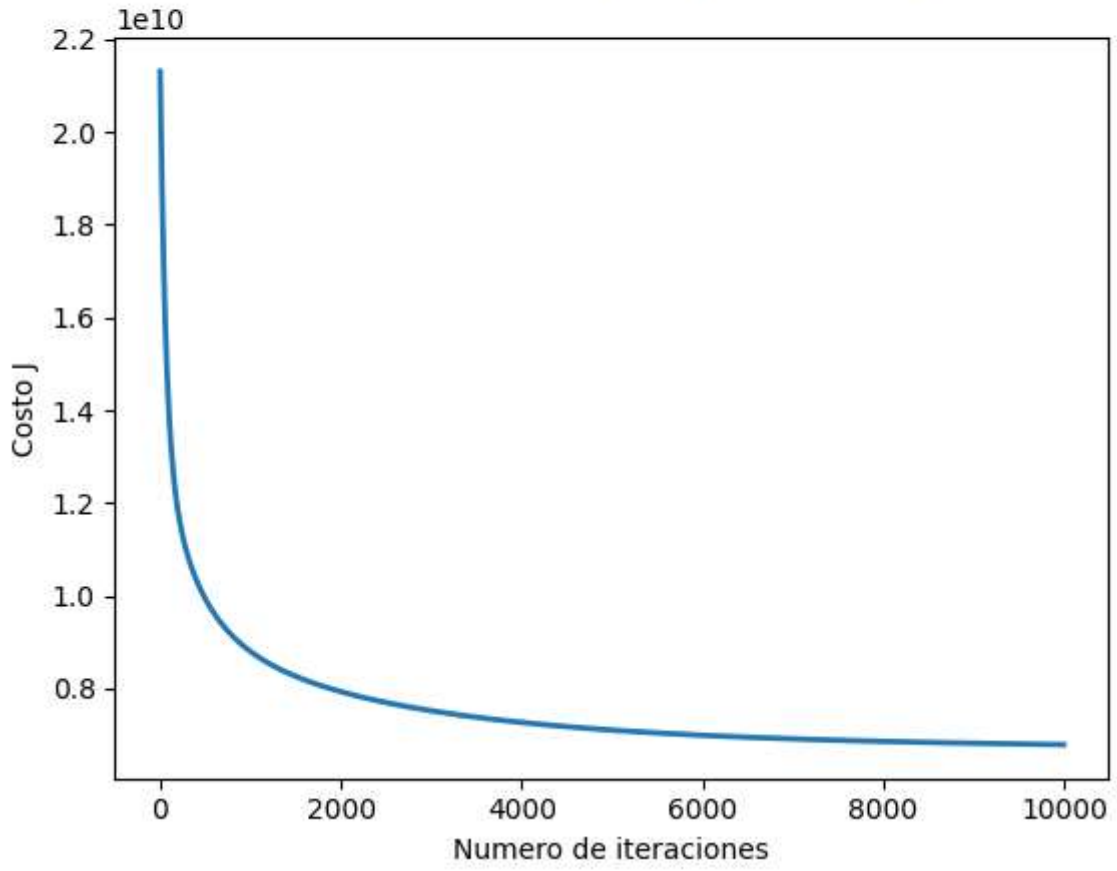
Predicción y Costo sin regularización:

```
theta calculado por el descenso por el gradiente: [ 64592.86884611  13065.34
 -27564.39038476 -39987.56969069  15976.37684252 -47001.62823354
  -5224.76410119  26099.56689808 101985.42658472  -3229.60868849
  -5839.38909603 -12503.31349146 -19554.59858747  10464.49401057
 123642.08955098  93912.13819736   8385.62313334  15006.99107868
 -61414.8345678 ]
El costo del Barco (usando el descenso por el gradiente) es: $-38186
```



Predicción y Costo con regularización:

```
theta calculado por el descenso por el gradiente: [ 64589.9509701   12776.76
   3468.87469083 -37008.57046459  13113.25373717 -28737.14249761
  -7836.6917663    4191.77864886  98997.38195035   8300.20890442
  -3077.66604924   5356.23128858 -34596.42468894  11994.83269798
  48700.99622208  74338.01871407   3948.7835791   29098.69673944
 -21438.50492587]
El costo del Barco (usando el descenso por el gradiente) es: $-31218
```



La ‘y’ predicha mejora significativamente pero no se logra un valor positivo a pesar de haber hecho varios intentos de prueba con diferentes valores de lambda_reg.

Con la ecuación de la normal:

No se logra una mejora de la ‘y’ predicha.

Sin Regularización:

```
Precio del Barco (usando la ecuación de la normal) es: $-26688
```

Con regularización:

```
Precio del Barco (usando la ecuación de la normal) es: $-28055
```

3. Aplicación de la Regularización a la Regresión Logística

Implementación de regresion logistica al dataset "WineQuality" para hacer la predicción de la calidad del vino. Preprocesamiento del dataset con Pandas.

3.1 Se aplicó la regularización a la función de Costo y a la Función de Descenso por el Gradiente

```
def calcularCosto(theta, X, y, lambda_):
    # Inicializar algunos valores utiles
    m = y.size  # numero de ejemplos de entrenamiento

    J = 0
    h = sigmoid(X.dot(theta.T))
    reg_term = (lambda_ / (2 * m)) * np.sum(np.square(theta[1:]))  # Término de regularización
    J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h))) + reg_term

    return J
```

```
def descensoGradiente(theta, X, y, alpha, num_iters, lambda_):
    # Inicializa algunos valores
    m = y.shape[0] # numero de ejemplos de entrenamiento

    # realiza una copia de theta, el cual será acutalizada por el descenso por el gradiente
    theta = theta.copy()
    J_history = []

    for i in range(num_iters):
        h = sigmoid(X.dot(theta.T))
        reg_term = (lambda_ / m) * theta  # Término de regularización
        reg_term[0] = 0  # No regularizar el término theta0
        theta = theta - (alpha / m) * ((h - y).dot(X) + reg_term)

        J_history.append(calcularCosto(theta, X, y, lambda_))
    return theta, J_history
```

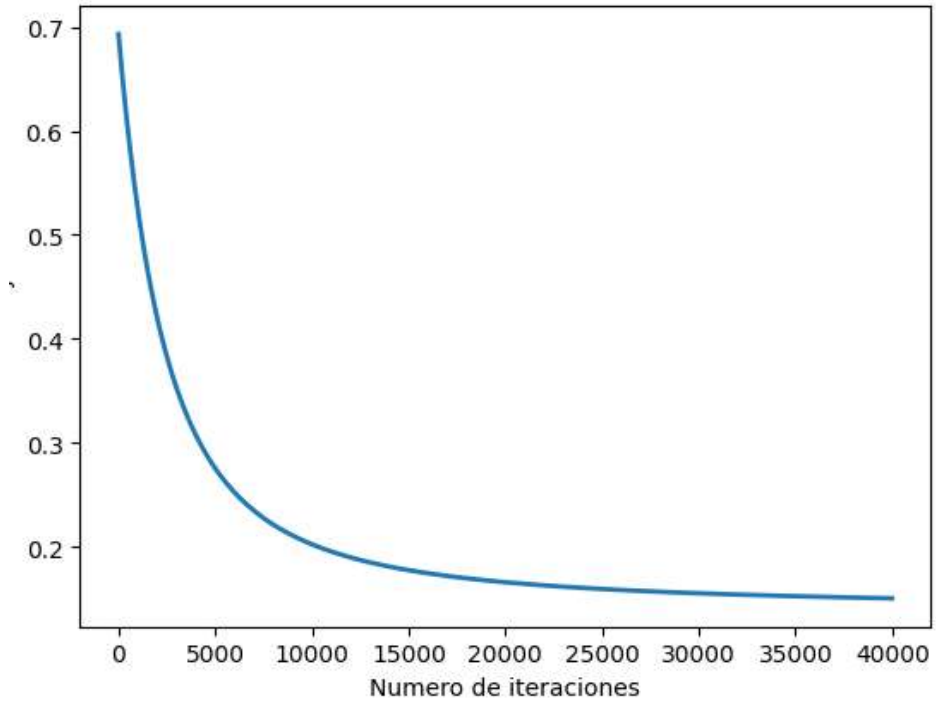
Para las predicciones se determinaron éstos valores como óptimos:

alpha = 0.001

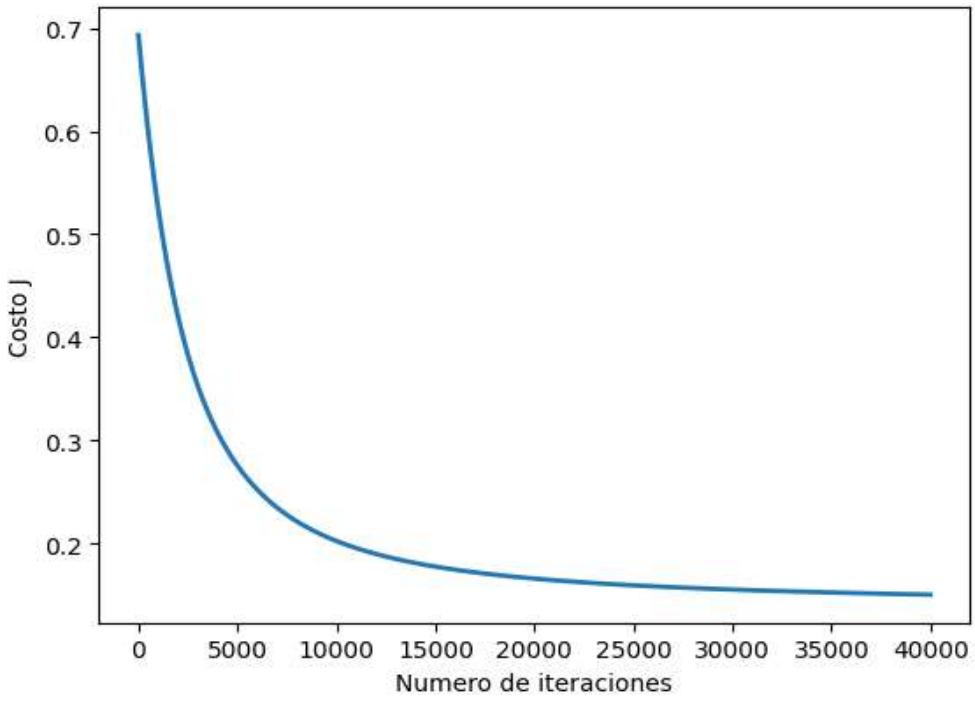
```
num_iters = 40000
lambda_ = 10
```

El gráfico de costo se mantiene en relación al costo sin regularización.

Costo con regularización:



Costo sin regularización:



La regularización en la regresión logística no ha influído en las predicciones.

Predición con Regularización:

Predicción para todo el conjunto de prueba

```
] predictions = []

  # Iterar sobre cada ejemplo en X_test_norm
  for example in X_test_norm:
      # Calcular la predicción para el ejemplo actual
      prediction = sigmoid(np.dot(example, theta))
      # Agregar la predicción a la lista de predicciones
      predictions.append(prediction)

  # Convertir la lista de predicciones a un arreglo de numpy
  predictions = np.array(predictions)

  # Imprimir las predicciones
  print(predictions)
```

```
[0.85097872 0.97472581 0.96292671 ... 0.95949059 0.96985068 0.95487837]
```

Predicción sin regularización:

```
# Imprimir las predicciones
print(predictions)
```

```
[0.85097872 0.97472581 0.96292671 ... 0.95949059 0.96985068 0.95487837]
```

Detalle de la aplicación de la Regularización en los archivos adjuntos.