Regularización en regresión lineal
Al momento de aplicar la regularización dentro del ejemplo de regresión lineal, se ha podido comprobar que los valores de theta se hacen cada vez mucho más pequeños, pero comparados con los mismos valores de alpha y número de iteraciones aplicadas dentro del modelo donde no se utilizó regularización.

Al momento de aplicar regularización es que se tuvo los siguientes valores base:

```
[173] alpha = 0.003
      num_iters = 10000
      theta_m = np.zeros(7)
      theta_m, J_history = gradientDescentMulti(X, y_multivariable , theta_m, alpha, num_iters)
```

Modelo donde no se utilizó regularización

Valores de theta

```
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta_m)))

theta calculado por el descenso por el gradiente: [ 1.74619928e+02 -1.78307582e-01  1.78616511e-01 -1.43430677e-03
  6.30868590e-03  4.73405552e+01  1.37516159e+02]
```

Predicción:

```
[197] pred = np.dot(X_test_norm, theta_m)
      pred_int = pred.astype(int)

      print(pred_int)
      print(Y_test_multivariable)

      print('Precision del conjuto de entrenamiento: {:.2f}%'.format(np.mean(pred_int == Y_test_multivariable) * 100))

      [327 250 214 ...  89  61  49]
      [327. 250. 214. ...  90.  61.  49.]
      Precision del conjuto de entrenamiento: 50.85%
```

Modelo donde se utilizó regularización

Esto se consiguió con un valor de lambda muy pequeño, 0,001

Valores de theta

```
[235] print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta_mr)))

      theta calculado por el descenso por el gradiente: [ 1.74619928e+02 -1.78296682e-01  1.78627391e-01 -1.45011959e-03
        6.31094144e-03  4.73405544e+01  1.37516097e+02]
```

Predicción:

Como podemos ver es que la diferencia es casi ninguna con el anterior modelo e incluso la precisión se ve afectada, por lo que cambiando los valores base obtenemos, valores distintos.

```
alpha = 0.03
num_iters = 6000
theta_l = np.zeros(19)
theta_l, J_history = descensoGradiente(theta_l, X_norm , y_train,  alpha, num_iters)
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')

print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta_l)))
```

Al cambiar los datos podemos ver que los datos van cerca del 50%, por lo que en este caso la regularización tiene un cambio mínimo dentro de si aplicarla o no.

Regularización en regresión logística

Para regularización logística nuevamente utilizaremos los datos de base de alfa y el número de iteraciones para ambos casos, haciendo variar solamente lambda

Modelo donde no se utilizó regularización

```
[387] def predict(theta, X):
          probabilities = sigmoid(np.dot(X, theta))
          return [1 if x >= 0.4 else 0 for x in probabilities]

[412] pred = predict(theta_l, X_test_log)

[413] print('Precision del conjuto de entrenamiento: {:.2f}%'.format(np.mean(pred == y_test) * 100))

      Precision del conjuto de entrenamiento: 59.61%
```

Modelo donde se utilizó regularización

con lambda igual a 1000, es decir que se utiliza un valor grande, para ayudar a los valores de theta a ser cada vez más pequeños

```
[428] pred = predict(theta_lr, X_test_log)

[429] print('Precision del conjuto de entrenamiento: {:.2f}%'.format(np.mean(pred == y_test) * 100))

      Precision del conjuto de entrenamiento: 61.47%
```

Podemos comprobar que al contrario de la regresión lineal, en este caso lambda si tiene efecto sobre theta, y ayuda que la precisión sea más alta dentro del modelo donde se utilizó regularización, con un valor de lambda muy grande, es decir que los datos se vuelven muy cercanos a 0, sin embargo la diferencia es muy pequeña, a medida de que lambda aumenta exponencialmente de 10 en 10, solo crece en 1% comparado con el anterior, respecto a precisión.