INFORME DE REGULARIZACIÓN

1. Introducción

La regularización es una técnica utilizada en el aprendizaje automático del machine learning para prevenir problemas en el modelo como ser el Subajuste o el sobreajuste (overfitting) de los modelos a los datos de entrenamiento. Porque cuando un modelo se ajusta demasiado a los datos de entrenamiento, el modelo se adapta totalmente a esos datos, lo que dificulta que el modelo generalice bien a nuevos datos dando así datos incorrectos.

Por ello se llevó el estudio de este método en los ejercicios previamente realizados y observar los cambios que el mismo pueda generar, así como su entendimiento.

La regularización es demasiado útil cuando existe sobreajuste de tal manera que se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. De esta manera el modelo se vuelve menos complejo.

Se compararán los resultados de los modelos regularizados con los no regularizados y se analizará su capacidad para prevenir el sobreajuste y mejorar el rendimiento en la generalización a nuevos datos.

De esta forma se sienta las bases para las pruebas y el uso de la regularización como una herramienta valiosa en el desarrollo de modelos de machine learning robustos y capaces de generalizar de manera efectiva.

2. Implementación de la regularización

La regularización será implementada en el código en los apartados que sean necesarios, desde la función de costo hasta el descenso por el gradiente de todas las funciones utilizadas en el código.

Además se la usará también en la ecuación de la normalización como una medida de comprobante en su impacto con los ejercicios hechos.

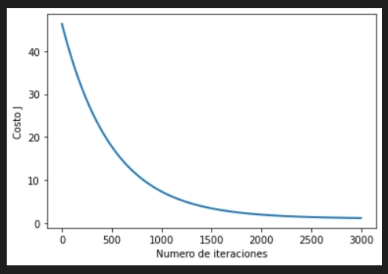
3. Ejemplos y resultados

- **Modelo de regresion lineal multivariable, una regresion polinomica y el calculo por la ecuacion de la normal**

```
# Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.001
num_iters = 3000
lambda_ = 1000
# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(7)
theta, J_history = gradientDescentMulti(X, y, theta, alpha, num_iters)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))

X_array = [1, 368.1,94.90,54.76,984.3832,0.20,119.40396]
X_array[1:7] = (X_array[1:7] - mu) / sigma

price = np.dot(X_array, theta)
print(format(price))
```

[97] ✓ 2.9s

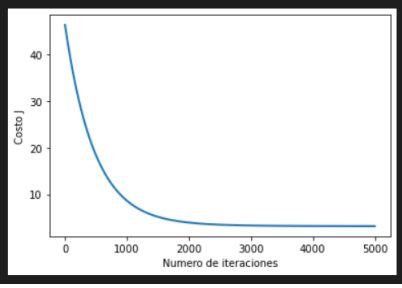
```
theta calculado por el descenso por el gradiente: [ 8.85779746  0.881115   -1.06715693  0.56354198 -1.20248531  0.15140943
 -0.05404995]
10.194620658030052
```




```
# Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.001
num_iters = 3000
lambda_ = 1000
# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(7)
theta, J_history = gradientDescentMultiWithReg(X, y, theta, alpha, num_iters, lambda_)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Numero de iteraciones')
pyplot.ylabel('Costo J')

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))

X_array = [1, 368.1,94.90,54.76,984.3832,0.20,119.40396]
X_array[1:7] = (X_array[1:7] - mu) / sigma

price = np.dot(X_array, theta)
print(format(price))
```

[96] ✓ 4.3s

```
theta calculado por el descenso por el gradiente: [ 8.83085437e+00  9.45383219e-01 -1.26378194e+00  4.52108657e-01
 -1.28679662e+00  1.27112869e-01 -3.55704878e-04]
10.178212969258889
```



```
# EJEMPLO de predicción
X_array = np.array([1,371.7, 96.00, 93.10, 983.6172, 0.03, 129.22797])


X_array[1:7] = (X_array[1:7] - mu) / sigma
price = np.dot(X_array, theta)
print(format(price))
```

✓ 0.0s

```
10.837258863666795
```

```
# Calcula los parametros con la ecuación de la normal
theta = normalEqn(X, y);

# Muestra los resultados optenidos a partir de la aplicación de la ecuación de la normal
print('Theta calculado a partir de la ecuación de la normal: {:s}'.format(str(theta)));

# Estimar el precio para una casa de superficie de 1650 sq-ft y tres dormitorios

X_array = [1, 368.1,94.90,54.76,984.3832,0.20,119.40396]
price = np.dot(X_array, theta)

print(format(price))
```

[220] ✓ 0.0s

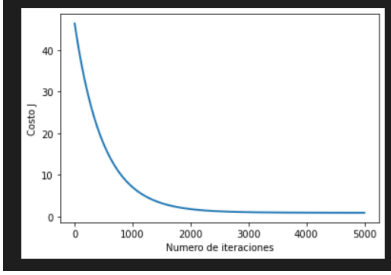
```
Theta calculado a partir de la ecuación de la normal: [ 1.24356448e+02  4.02527091e-02 -2.00020733e-01  9.64512424e-04
 -1.11334316e-01  2.48292631e-01  1.03671622e-03]
10.82213546194857
```

```
# Calcula los parametros con la ecuación de la normal regularizada
lambda_ = 1000
theta = normalEqn(X, y , lambda_);

# Muestra los resultados optenidos a partir de la aplicación de la ecuación de la normal
print('Theta calculado a partir de la ecuación de la normal: {:s}'.format(str(theta)));

# Estimar el precio para una casa de superficie de 1650 sq-ft y tres dormitorios

X_array = [1, 368.1,94.90,54.76,984.3832,0.20,119.40396]
price = np.dot(X_array, theta)

print(format(price))
```

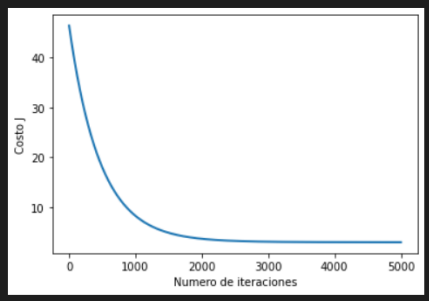
[222] ✓ 0.0s

```
Theta calculado a partir de la ecuación de la normal: [ 0.10859633  0.07260536 -0.19753803 -0.00083656  0.00272476  0.31731712
  0.00114019]
10.924274019158702
```

```
theta calculado por el descenso por el gradiente: [ 9.25852674  0.60465817 -0.84800079  0.24110515 -0.7149602   0.00935186
  0.21028656  0.61338929 -0.92696101 -0.01224606 -0.71509319  0.10577233
 -0.16997119]
10.77841322149414
```

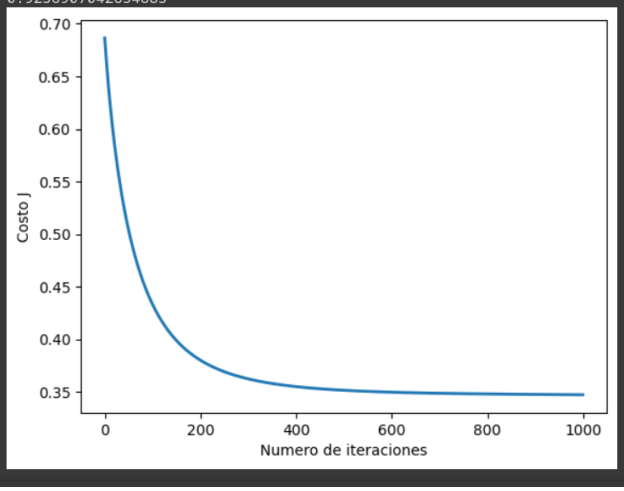


```
theta calculado por el descenso por el gradiente: [ 8.83085437  0.57379948 -0.79187829  0.24780039 -0.6873812   0.01861638
  0.18080665  0.5823523  -0.86079459  0.01667909 -0.68747021  0.10094797
 -0.16433786]
10.301762938436926
```



- **En la clasificación logística**

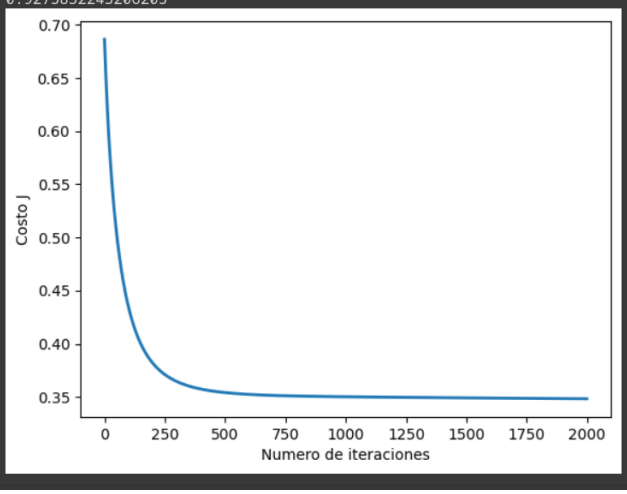
```
theta calculado por el descenso por el gradiente: [ 5.17346581e-02  2.48079414e-02 -1.05661650e-01 -2.58213218e-02
  6.48434016e-02  3.07292950e-01 -6.33940905e-03 -5.31192255e-03
  1.58505447e-03 -2.39513177e-03 -1.42458259e-02 -3.81320708e-03
 -1.14097545e-02  1.36172416e-04  8.93399127e-02]
0.9256907042654885
```




```
[51] print('Precisión de entrenamiento: {:.2f} %'.format(np.mean(y_umbral == y_test[0]) * 100))

     Precisión de entrenamiento: 62.40 %
```

- **Regularizado**

```
theta calculado por el descenso por el gradiente: [ 0.06415505  0.02680103 -0.10627686 -0.02719871  0.08974087  0.30542082
 -0.01322674 -0.01093201 -0.00132322 -0.00804726 -0.03071376 -0.00947611
 -0.02105441 -0.01310506  0.10447743]
0.9273832243206205
```




```
[156] print('Precisión de entrenamiento: {:.2f} %'.format(np.mean(y_umbral == y_test[0]) * 100))

      Precisión de entrenamiento: 63.52 %
```

- **En clasificación multiclase**
- **sin regularización**

```
pred_test = predictOneVsAll(all_theta, X_norm)
print('Precision del conjunto de entrenamiento: {:.2f}%'.format(np.mean(pred_test == y_train) * 100))
```
[107] ✓ 0.0s Python

```
Precision del conjunto de entrenamiento: 71.07%
```

```
X_test1 = X_test.copy()
X_test = (X_test1 - mu) / sigma
pred_train = predictOneVsAll(all_theta, X_test1)
print('Precision del conjunto de prueba: {:.2f}%'.format(np.mean(pred_train == y_test) * 100))
```
[111] ✓ 0.0s Python

```
Precision del conjunto de prueba: 69.46%
```

- **Con regularización**

```
pred_test = predictOneVsAll(all_theta, X_norm)
print('Precision del conjunto de entrenamiento: {:.2f}%'.format(np.mean(pred_test == y_train) * 100))
```
[140] ✓ 0.0s

```
Precision del conjunto de entrenamiento: 71.08%
```

```
X_test1 = X_test.copy()
X_test = (X_test1 - mu) / sigma
pred_train = predictOneVsAll(all_theta, X_test1)
print('Precision del conjunto de prueba: {:.2f}%'.format(np.mean(pred_train == y_test) * 100))
```
[141] ✓ 0.0s

```
Precision del conjunto de prueba: 71.31%
```

4. Conclusiones

- No hubo un impacto significativo en la máquina. Esto sugiere que la implementación de la regularización no implicó una carga computacional significativa en los recursos del sistema.

- Es importante, ya que demuestra que la regularización es una técnica eficiente y escalable, que puede ser aplicada sin comprometer el rendimiento general de la máquina y de hecho puede mejorar en casos donde se tengan varias características.

- Al estar la regularización enfocada en el sobreajuste (overfitting) de los modelos, se vió predicciones más precisas en datos no vistos durante el entrenamiento.

- La regularización puede ser aplicada en diferentes tipos de modelos de aprendizaje automático, como se observó en los ejemplos mencionados donde se usó en polinómica, multivariable, multiclase y otros.

- El hecho de que se haya observado una mejora más significativa en el caso de la clasificación multiclase con los datos de prueba resalta la importancia de la regularización en problemas más complejos como en el dataset con el que se trabajó en ese ejemplo.

- La regularización ayuda a mantener un equilibrio adecuado entre el ajuste a los datos de entrenamiento y la capacidad de generalización a nuevos datos.

- Aunque la mejora fue mínima en otros ejemplos como la regresión logística, polinómica y multivariable, es importante destacar que incluso una pequeña mejora puede ser valiosa en ciertos casos.

- También otro de los objetivos del mismo es obtener modelos más simples e interpretables, lo cual puede ser beneficioso dependiendo del objetivo del análisis.