Universitario: Ceron Beimar Miguel

Carrera: Ingenieria de Sistemas

```
#Datashet de Kaggle:
#https://www.kaggle.com/datasets/vuppalaadithyasairam/kidney-stone-prediction-based-on-urine-analysis?select=kindey+stone+

# Repositorio GITHub:
# https://github.com/Beimar98/SIS420/tree/main/2do%20parcial%20IA



# Datos:

#0 1 2 3 4 5  ---> para X (gravity  ph  osmo  cond  urea  calc)
#            6 ---> para Y (target) -->0- ausencia de piedra 1- presencia de piedra(via urinaria en obstrucción ?)

#X ,y = data[:, 0:6], data[:, 6]

# el numero de iteraciones suficientes para alcanzar los valores ideales de theta: num_iters = 500
# el mejor valor de alfa (coheficiente de aprendizaje) : alpha = 0.00001
#########################################################################################################################################
#Si la probabilidad es menor a cero o menor a 0.50 (No esta obstruida)
#Si la probabilidad es mayor a 0.50(esta obstruida)

#probabilidad de que la via urinaria este obstruida :0.5142566518149368
#Precisión de entrenamiento: 51.96 %
#########################################################################################
```

Ejercicion de programación - Regresión Logistica

En este ejercicio se implementa regresion logistica y se aplica a dos diferentes datasets.

```
# se utiliza para el manejo de rutas y directorios.
import os

# Calculo cientifico y vectorial para python
import numpy as np

# Librerias para graficar
from matplotlib import pyplot

# Modulo de optimización de scipy
from scipy import optimize

# le dice a matplotlib que incruste gráficos en el cuaderno
%matplotlib inline
```

1 Regresion Logistica

En esta parte del ejercicio, creará un modelo de regresión logística para predecir si un estudiante será admitido en una universidad. Suponga que es el administrador de un departamento universitario y desea determinar las posibilidades de admisión de cada solicitante en función de sus resultados en dos exámenes. Tiene datos históricos de solicitantes anteriores que puede usar como un conjunto de capacitación para la regresión logística. Para cada ejemplo de capacitación, se tiene las calificaciones del solicitante en dos exámenes y la decisión de admisión. Su tarea es crear un modelo de clasificación que calcule la probabilidad de admisión de un solicitante en función de los puntajes de esos dos exámenes.

La siguiente celda cargará los datos y las etiquetas correspondientes:

```
# Cargar datos
# Las dos primeras columnas contienen la nota de dos examenes y la tercera columna
# contiene la etiqueta que indica si el alumno ingreso o no a la universidad.
data = np.loadtxt(os.path.join('/content/zkindey.csv'), delimiter=',')
X, y = data[:, 0:6], data[:, 6]
print(X)
print(y)
```

```
[[  1.021   4.91  725.     14.    443.      2.45 ]
 [  1.017   5.74  577.     20.    296.      4.49 ]
 [  1.008   7.2   321.     14.9   101.      2.36 ]
```

```
...
[  1.022   4.52  488.     37.1   198.      4.3  ]
[  1.033   7.43  720.      8.8   158.      0.82 ]
[  1.027   3.75  558.     21.6   607.      1.96 ]]
[0. 0. 0. ... 1. 0. 1.]
```

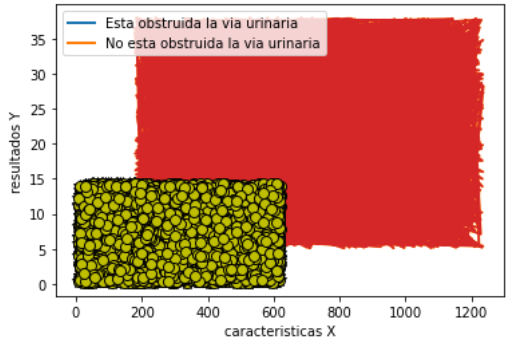
1.1 Visualizar los datos

Antes de comenzar a implementar cualquier algoritmo de aprendizaje, siempre es bueno visualizar los datos si es posible. Mostramos los datos en una gráfica bidimensional llamando a la función `plotData`. Se completará el código en `plotData` para que muestre una figura donde los ejes son los dos puntajes de los dos examenes, los ejemplos positivos y negativos se muestran con diferentes marcadores.

```
def plotData(X, y):
    # Gragica los puntos de datos X y y en una nueva figura. Grafica los puntos de datos con * para los positivos y
    # o para los negativos.

    # Crea una nueva figura
    fig = pyplot.figure()

    # Find Indices of Positive and Negative Examples
    pos = y == 1
    neg = y == 0

    # Plot Examples
    pyplot.plot(X[pos, 0], X[pos, 1],X[pos, 2],X[pos, 3],X[pos, 4],X[pos, 5], 'k*', lw=2, ms=10)
    pyplot.plot(X[neg, 0], X[neg, 1],X[pos, 2],X[pos, 3],X[pos, 4],X[pos, 5], 'ko', mfc='y', ms=8, mec='k', mew=1)
```

Se llama a la función implementada para mostrar los datos cargados:

```
plotData(X, y)
# adiciona etiquetas para los ejes
pyplot.xlabel('caracteristicas X')
pyplot.ylabel('resultados Y')
pyplot.legend(['Esta obstruida la via urinaria', 'No esta obstruida la via urinaria'])
pass
```



1.2 Implementacion

1.2.1 Fución Sigmoidea

La hipotesis para la regresión logistica se define como:

$$h_\theta(x) = g(\theta^T x)$$

donde la función g is la función sigmoidea. La función sigmoidea se define como:

$$g(z) = \frac{1}{1+e^{-z}}$$

.

Los resultados que debe generar la funcion sigmoidea para valores positivos amplios de `x`, deben ser cercanos a 1, mientras que para valores negativos grandes, la sigmoide debe generar valores cercanos 0. La evaluacion de `sigmoid(0)` debe dar un resultado exacto de 0.5. Esta funcion tambien debe poder trabajar con vectores y matrices.

Haz doble clic (o pulsa Intro) para editar

```
def sigmoid(z):
    # Calcula la sigmoide de una entrada z
    # convierte la intrada a un arreglo numpy
    z = np.array(z)
```

```
    g = np.zeros(z.shape)

    g = 1 / (1 + np.exp(-z))

    return g
```

Se calcula el valor de la sigmoide aplicando la funcion sigmoid con `z=0`, se debe obtener un resultado de 0.5. RE recomienda experimentar con otros valores de `z`.

```
# Prueba la implementacion de la funcion sigmoid
z = 0
g = sigmoid(z)

print('g(', z, ') = ', g)
```

```
 g( 0 ) =  0.5
```

1.2.2 Función de Costo y Gradiente

Se implementa la funcion cost y gradient, para la regresión logistica. Antes de continuar es importante agregar el termino de intercepcion a X.

```
# Configurar la matriz adecuadamente, y agregar una columna de unos que corresponde al termino de intercepción.
m, n = X.shape
# Agraga el termino de intercepción a A
X = np.concatenate([np.ones((m, 1)), X], axis=1)
```

La funcion de costo en una regresión logistica es:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

y el gradiente del costo es un vector de la misma longitud como θ donde el elemento j^{th} (para $j = 0, 1, \cdots, n$) se define como:

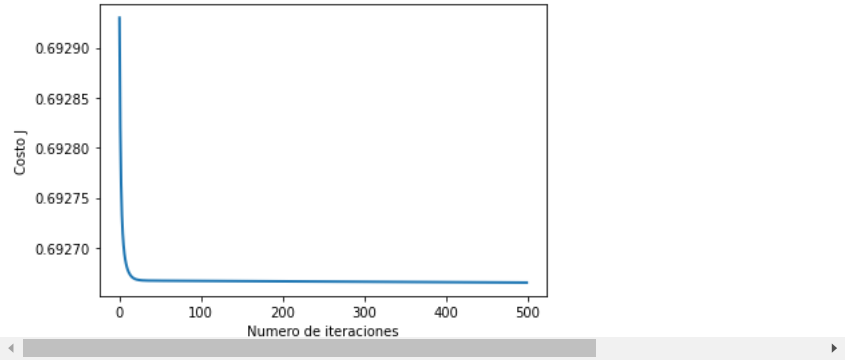
$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Si bien este gradiente parece idéntico al gradiente de regresión lineal, la fórmula es diferente porque la regresión lineal y logística tienen diferentes definiciones de $h_\theta(x)$.

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

    J = 0
    h = sigmoid(X.dot(theta.T))
    J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))

    return J


def descensoGradiente(theta, X, y, alpha, num_iters):
    # 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))
        theta = theta - (alpha / m) * (h - y).dot(X)

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


# Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.00001
num_iters = 500

# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(7)
theta, J_history = descensoGradiente(theta, X, y, 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')
print("costo min: "+ str(J_history[-1]) )

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

# verificar si ingresa o no a la universidad
X_array = [1,1.021,4.91,725,14,443,2.45]
aprueba = sigmoid(np.dot(X_array, theta))   # Se debe cambiar esto

print('probabilidad de usar  rapamicina  (usando el descenso por el gradiente):{:.0f}'.format(aprueba))
print("Si la probabilidad es menor a cero o menor a 0.50 (No esta obstruida la via urinaria) \n Si la probabilidad es mayor a 0.50(esta
```

```
costo min: 0.6926653178503449
theta calculado por el descenso por el gradiente: [-1.54043728e-06 -1.40734220e-06
  1.85762721e-05  1.62214220e-04  9.18094411e-05]
probabilidad de usar  rapamicina  (usando el descenso por el gradiente):1
Si la probabilidad es menor a cero o menor a 0.50 (No esta obstruida la via urinar
 Si la probabilidad es mayor a 0.50(esta obstruida la via urinaria)
```




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

    J = 0
    grad = np.zeros(theta.shape)

    h = sigmoid(X.dot(theta.T))

    J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))
    grad = (1 / m) * (h - y).dot(X)

    return J, grad
```

Se prueba la funcion `costFunction` utilizando dos casos de prueba para θ.

```
# Inicializacion de parametros de ajuste
initial_theta = np.zeros(n+1)
print(initial_theta)
cost, grad = costFunction(initial_theta, X, y)

print('Costo en theta inicial (zeros): {:.3f}'.format(cost))
print('Costo esperado (aproximado): 0.693\n')
print(grad)
print('Gradiente en theta inicial (zeros):')
print('\t[{:.4f}, {:.4f}, {:.4f}]'.format(*grad))
print('Gradiente esperado (aproximado):\n\t[-0.1000, -12.0092, -11.2628]\n')
```

```
[0. 0. 0. 0. 0. 0. 0.]
Costo en theta inicial (zeros): 0.693
Costo esperado (aproximado): 0.693

[-0.01338413 -0.01372001 -0.06727775 -9.74516413 -0.29746483 -5.58890191
 -0.11784095]
Gradiente en theta inicial (zeros):
        [-0.0134, -0.0137, -0.0673]
Gradiente esperado (aproximado):
        [-0.1000, -12.0092, -11.2628]
```

```
# Calcula y muestra el costo y el gradiente con valores de theta diferentes a cero

test_theta = np.array([-24,1.021,4.91,725,14,443,2.45])
#test_theta = np.array([-11.74749157, 0.09927308, 0.09316497])
```

```
print(test_theta)
cost, grad = costFunction(test_theta, X, y)

print('Costo en theta prueba: {:.3f}'.format(cost))
print('Costo esperado (aproximado): 0.218\n')

print('Gradiente en theta prueba:')
print('\t[{:.3f}, {:.3f}, {:.3f}]'.format(*grad))
print('Gradiente esperado (aproximado):\n\t[0.043, 2.566, 2.647]')
```

```
    [-24.      1.021   4.91  725.     14.    443.      2.45 ]
    Costo en theta prueba: nan
    Costo esperado (aproximado): 0.218

    Gradiente en theta prueba:
            [0.487, 0.498, 2.709]
    Gradiente esperado (aproximado):
            [0.043, 2.566, 2.647]
    <ipython-input-14-1a3084f56ce3>:10: RuntimeWarning: divide by zero encountered in log
      J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))
```

1.2.3 Parámetros de aprendizaje usando `scipy.optimize`

En el codigo anterior se encontró los parámetros óptimos de un modelo de regresión lineal al implementar el descenso de gradiente. Se implemento una función de costo y se calculó su gradiente, utilizando el algoritmo del descenso por el gradiente.

En lugar de realizar los pasos del descenso por el gradiente, se utilizará el [módulo `scipy.optimize`] (https://docs.scipy.org/doc/scipy/reference/optimize.html). SciPy es una biblioteca de computación numérica para `python`. Proporciona un módulo de optimización para la búsqueda y minimización de raíces. A partir de `scipy 1.0`, la función `scipy.optimize.minimize` es el método a utilizar para problemas de optimización (tanto restringidos como no restringidos).

For logistic regression, you want to optimize the cost function $J(\theta)$ with parameters θ. Concretely, you are going to use `optimize.minimize` to find the best parameters θ for the logistic regression cost function, given a fixed dataset (of X and y values). You will pass to `optimize.minimize` the following inputs:

Para la regresión logística, se desea optimizar la función de costo $J(\theta)$ con los parámetros θ. Concretamente, se va a utilizar `optimize.minimize` para encontrar los mejores parámetros θ para la función de costo de regresión logística, dado un dataset fijo (de valores X e y). Se pasara a `optimize.minimize` las siguientes entradas:

- `costFunction`: Una función de costo que, cuando se le da el dataset de entrenamiento y un θ particular, calcula el costo de regresión logística y el gradiente con respecto a θ para el dataset(X, y). Es importante tener en cuenta que solo se pasa el nombre de la función sin el paréntesis. Esto indica que solo proporcionamos una referencia a esta función y no evaluamos el resultado de esta función.
- `initial_theta`: Los valores iniciales de los parametros que se tratan de optimizar.
- `(X, y)`: Estos son argumentos adicionales a la funcion de costo.
- `jac`: Indicación si la función de costo devuelve el jacobiano (gradiente) junto con el valor de costo. (True)
- `method`: Método / algoritmo de optimización a utilizar
- `options`: Opciones adicionales que pueden ser específicas del método de optimización específico. Solo se indica al algoritmo el número máximo de iteraciones antes de que termine.

Si se ha completado la `costFunction` correctamente, `optimize.minimize` convergerá en los parámetros de optimización correctos y devolverá los valores finales del costo y θ en un objeto de clase.

Al usar `optimize.minimize`, no se tuvo que escribir ningún bucle ni establecer una tasa de aprendizaje como se hizo para el descenso de gradientes. Todo esto se hace mediante `optimize.minimize`: solo se necesita proporcionar una función que calcule el costo y el gradiente.

A continuación, se tiene el código para llamar a `optimize.minimize` con los argumentos correctos.

```
# Establecer las opciones para optimize.minimize
options= {'maxiter': 1000}

# revisar la documentacion de scipy's optimize.minimize para mayor descripcion de los parametros
# La funcion devuekve un objeto `OptimizeResult`
# Se utiliza el algoritmo de Newton truncado para la optimización.
res = optimize.minimize(costFunction,
                        initial_theta,
                        (X, y),
                        jac=True,
                        method='TNC',
                        options=options)

# la propiedad fun del objeto devuelto por `OptimizeResult`
# contiene el valor del costFunction de un theta optimizado
cost = res.fun

# Theta optimizada esta en la propiedad x
theta = res.x

# Imprimir theta en la pantalla
```

```
print('Costo con un valor de theta encontrado por optimize.minimize: {:.3f}'.format(cost))
print('Costo esperado (aproximado): 0.203\n');

print('theta:')
print('\t[{:.3f}, {:.3f}, {:.3f}]'.format(*theta))
print('Theta esperado (aproximado):\n\t[-25.161, 0.206, 0.201]')
```

```
Costo con un valor de theta encontrado por optimize.minimize: 0.693
Costo esperado (aproximado): 0.203

theta:
        [-1.305, 1.304, -0.015]
Theta esperado (aproximado):
        [-25.161, 0.206, 0.201]
```

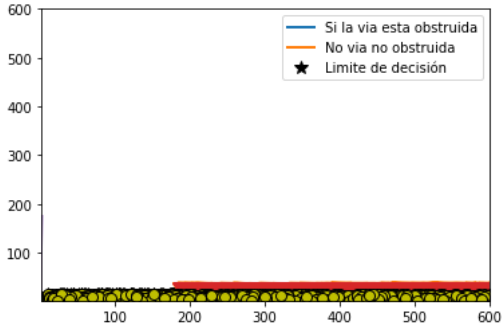
Una vez que se completa `optimize.minimize`, se usa el valor final de θ para visualizar el límite de decisión en los datos de entrenamiento.

Para hacerlo, se implementa la función `plotDecisionBoundary` para trazar el límite de decisión sobre los datos de entrenamiento.

```
def plotDecisionBoundary(plotData, theta, X, y):
    """
    Grafica los puntos X y Y en una nueva figura con un limite de desicion definido por theta.
    the data points X and y into a new figure with the decision boundary defined by theta.
    Grafica los puntos con * para los ejemplos positivos y con o para los ejemplos negativos.

    Parametros:
    ----------
    plotData : func
        A function reference for plotting the X, y data.

    theta : array_like
        Parametros para la regresion logistica. Un vecto de la forma (n+1, ).

    X : array_like
        Data set de entrada. Se supone que X es una de las siguientes:
            1) Matriz Mx3, donde la primera columna es una columna de unos para intercepción.
            2) Matriz MxN, N> 3, donde la primera columna son todos unos.

    y : array_like
        Vector de datos de etiquetas de la forma (m, ).
    """
    # hacer que theta sera un arreglo numpy
    theta = np.array(theta)

    # Graficar los datos (recordar que la primera columna en X es la intercepción)
    plotData(X[:, 1:7], y)

    if X.shape[1] <= 7:
        # Solo se requieren 2 puntos para definir una linea, para lo cual se eligen dos puntos finales
        plot_x = np.array([np.min(X[:, 1]) - 2, np.max(X[:, 1]) + 2])

        # Calcular la línea límite de decisión
        plot_y = (-1. / theta[2]) * (theta[1] * plot_x + theta[0])

        print(plot_x)
        print(plot_y)
        # Graficar y ajustar los ejes para una mejor visualización
        pyplot.plot(plot_x, plot_y)

        # Leyenda, especifica para el ejercicio
        pyplot.legend(['Si la via esta obstruida', 'No via no obstruida', 'Limite de decisión'])
        pyplot.xlim([2, 600])
        pyplot.ylim([2, 600])
    else:
        # Rango de la grilla
        u = np.linspace(-1, 1.5, 50)
        v = np.linspace(-1, 1.5, 50)

        z = np.zeros((u.size, v.size))
        # Evalua z = theta*x sobre la grilla
        for i, ui in enumerate(u):
            for j, vj in enumerate(v):
                z[i, j] = np.dot(np.mapFeature(ui, vj), theta)

        z = z.T  # importante transponer z antes de llamar al contorno
        # print(z)

        # Plot z = 0
        pyplot.contour(u, v, z, levels=[0], linewidths=2, colors='g')
        pyplot.contourf(u, v, z, levels=[np.min(z), 0, np.max(z)], cmap='Greens', alpha=0.4)
```

```
# Graficar limites
plotDecisionBoundary(plotData, theta, X, y)
```

```
[-0.995  3.04 ]
[-171.63937837  175.35076953]
```



1.2.4 Evaluación de la regresión logistica

Después de aprender los parámetros, se puede usar el modelo para predecir si un estudiante en particular será admitido. Para un estudiante con una puntuación en el Examen 1 de 45 y una puntuación en el Examen 2 de 85, debe esperar ver una probabilidad de admisión de 0,776. Otra forma de evaluar la calidad de los parámetros que hemos encontrado es ver qué tan bien predice el modelo aprendido en nuestro conjunto de entrenamiento.

```
def predict(theta, X):
    """
    Predecir si la etiqueta es 0 o 1 mediante regresión logística aprendida.
    Calcula las predicciones para X usando un umbral en 0.5 (es decir, si sigmoide (theta.T * x)> = 0.5, predice 1)

    Parametros
    ----------
    theta : array_like
        Parametros para regresion logistica. Un vecto de la forma (n+1, ).

    X : array_like
        Datos utilizados para el calculo de las predicciones.
        La fila es el numero de los puntos para calcular las predicciones,
        y las columnas con el numero de caracteristicas.

    Devuelve
    -------
    p : array_like
        Predicciones y 0 o 1 para cada fila en X.
    """
    m = X.shape[0] # Numero de ejemplo de entrenamiento

    p = np.zeros(m)

    p = np.round(sigmoid(X.dot(theta.T)))
    return p
```

Una vez entrenado el modelo se procede a realizar la prediccion y evaluación de los resultados de predecir cual es el valor que vota el modelo para todos los datos utilizados en el entrenamiento.

```
#  Predice la probabilidad de ingreso para un estudiante con nota de 45 en el examen 1 y nota de 85 en el examen 2
prob = sigmoid(np.dot([1, 1.021,4.91,725,14,443,2.45], theta))
#print('probabilidad de que un empleado sea dejo o abandono el area o puesto es del : {:.3f}%'.format(prob))
print("Si la probabilidad es menor a cero o menor a 0.50 (No esta obstruida) \n Si la probabilidad es mayor a 0.50(esta obstruida) ")
print("\nprobabilidad de que la via urinaria este obstruida :"+str( prob ))
# Compute accuracy on our training set
p = predict(theta, X)
print('Precisión de entrenamiento: {:.2f} %'.format(np.mean(p == y) * 100))
```

```
Si la probabilidad es menor a cero o menor a 0.50 (No esta obstruida)
 Si la probabilidad es mayor a 0.50(esta obstruida)

probabilidad de que la via urinaria este obstruida :0.5142566518149368
Precisión de entrenamiento: 51.96 %
```

✓ 0 s completado a las 20:53