Metodod del gradiente y regresión lineal

En esta unidad vamos a implementar una regresión lineal de una variable utilizando el método del gradiente y vamos a compara el resultado obtenido con el resultado calculado con scipy.. Antes de comenzar asegúrese que los siguientes archivos se encuentren en su carpeta:

- func.py
- met_grad.py
- plot_linreg.py
- utli_comuna.txt

El script met_grad.py sera el programa principal desde el cual va llamar las funciones que implementan las distintas operaciones. Esas funciones las va a escribir en el programa func.py.

Suponga que usted es el gerente de una franquicia de carritos de sopaipillas, y está considerando varias comunas para instalar un carrito nuevo. La empresa ya tiene carritos en varias comunas y usted tiene datos de ganancias y población de esas ciudades. A usted le gustaría utilizar esos datos para decidir la ubicación de la próxima filial. El archivo util_comuna.txt contiene el set de datos para la regresión lineal. La primera columna es la población de la comuna y la segunda columna es la utilidad del carrito de sopaipillas.

1. Visualizar la data

Con el fin de obtener una mejor idea sobre el set de datos disponibles, puede ser de gran utlidad visualizar los datos. Afortunadamente ayer aprendimos a usar una librería python para visualizar data que es Matplotlib. Como vamos a hacer una regresión lineal de una variable, es posible visualizarla mediante un scatter plot en 2D.

Escriba una función en func.py para visualizar la data contenida en el archivos util_coumuna.txt. Utilice la libreria de visualización de datos Matplotlib. Notese que la función toma como argumento dos dos arrays de numpy que son X (corresponde a las poblaciones de la ciudad) e Y (corresponde a las utilidades).

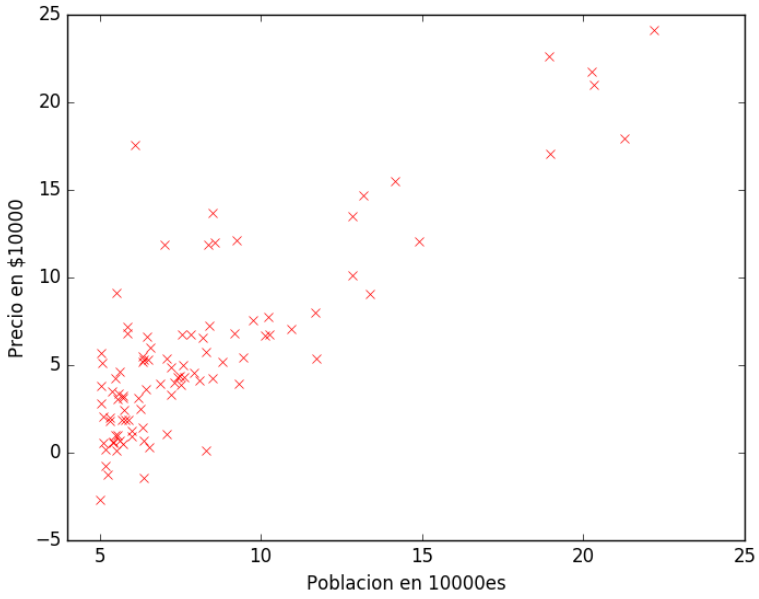

Figura 1: Visualización de los datos.

Nótese que las primeras lineas del script son las siguientes:

```
util_pob = np.genfromtxt(sys.argv[1], delimiter=',', names=['x', 'y'])

#Preprocesamiento de los datos
Y = util_pob['y']
m = len(Y)
Y.shape = (m,1)
X_org = util_pob['x']
X_org.shape = (m,1)

#Visualizar los datos:
plot_data(X_org,Y)
```

Aquí se extraen los datos y se guardan en un array llamado *util_pop*. Finalmente se extraen los datos de X y de Y del array con los datos. La última linea del fragmento de código llama a la función plot_data.py, alimentada con datos de población y ganancia contenidos en X_org e Y respectivamente.

2. Método del gradiente

Una vez que se haya familliarizado con los datos su misión es ajustar los parámetros de la regresión lineal θ a nuestro set de datos, utilizando el método del gradiente. Las funciones con las ecuaciones las implementará en el archivo func.py. Estas funciones las llamará desde met_grad.py para correr el programa. Antes de implementar las ecuaciones del método del gradiente, debemos definir los parámetros fundamentales del sistema que son el largo del set de datos y el array con los parámetros a ajustar, θ.

```
#El número de puntos en el set de data
m = len(Y)
#Inicializar la matrix theta
theta = np.zeros(( 2,1))
#insertar una columna de unos al comienzo del array X
X = np.insert(X_org,0,1,axis=1)
```

El último paso es agregar una columna de 1.0s al array X, debido a que el parámetro θ_0 siempre va multiplicado con un 1.0.

2.1. Función de costo

La función objetivo de la regresión lineal, es decir la función que debe minimizar para obtener los parámetros óptimos es:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

La función hipótesis está dada por:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 \tag{2}$$

Recuerdo que los parámetros de su modelo son los valores θ_J. Eso son los valores que debe ajustar para minimizar $J(\theta)$. $x^{(i)}$ corresponde a un vector columna del tipo $(1, x_i)$, en python corresponde a $X[i]$.

Escriba una función de python que toma como argumento un array X y un array Y y calcula la función de costo $J(\theta)$. Para correr el programa llame la función desde met_grad.py y corra el script.

Tips: Una sumatoria se puede calcular acumulando el valor sobre un float previamente iniciado. Por ejemplo, la sumatoria $\sum_{i=1}^{5} i^2$ se puede calcular como:

```
s = 0.0
for j in range(1,5):
   s = s + j**2
print s
>> 55
```

Además recuerdo que X e Y no son escalares, sino matrices que guardan las datos que estamos analizando. Si quiere acceder al miembor ij de la matriz M, lo debe escribir: M[i][j] donde i corresponde a la fila e j a la columna. M[i,:] le devuelve un vector con la i-esima fila y M[:,j] devuelve un vector con la j-esima columna El valor inicial de la función de costo debe ser 32.07.

2.2. Decender a lo largo del gradiente

Utilizaremos un método del gradiente llamado batch gradient decent. En este método cada en iteración se realiza una actualización del gradiente según la ecuación:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3}$$

Como tenemos un problema de una variable debes actualizar dos parámetros, θ_0 y θ_1. Por lo tanto las dos ecuaciones son:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \tag{4}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \tag{5}$$

Escriba una función que actualice los gradientes hasta que se alcance la convergencia Escriba esta función en el modulo func.py y llame la función desde met_grad.py. Verifique la convergencia calculando la función de costo en cada paso de la iteración.

3. Comparar con scipy

Compare el resultado obtenido con el resultado que le entrega la rutina de regresión lineal *lineregress* en la libreria scypy.stats. ¿Cuantos ciclos de iteración se requieren para llegar a el mismo resultado en la tercera décimal? Finalmente para poder visualizar su regresión lineal puede remover el comentario de la la última linea de met_grad.py:

```
#plot_linreg(X[:,1],Y[:,0],a,b)
```

Y hacer andar nuevamente el programa. Felicitaciones por su primer programa de regresión lineal.