Rubén Sánchez Mayén A01378379

Algoritmo Manual

Se implementó un algoritmo que calcula un modelo de **regresión lineal múltiple** que se adecué a las muestras presentadas, utilizando el **método del gradiente**.

Muestras

En el mismo archivo dónde se encuentra la implementación está la función ***generate_samples(m)***, que genera un ***m*** número de muestras, colocando los valores de las **variables independientes** en una **matriz**, los valores de la **variable dependiente** en un **arreglo** y regresando ambas en un ***return.***

La función ***generate_samples(m)*** por defecto representa la siguiente función lineal:

$$f(x) = 2x_1 + 3x_2 + \frac{9}{2}x_3 + 10$$

Para conseguir un **modelo imperfecto** se suma una cantidad aleatoria de **ruido** a cada valor generado de la **variable dependiente**. **El ruido** es número decimal aleatorio en el **rango (-2,2)** incluyendo ambos límites. Cada llamada a la función crea un **set de muestras similar**, pero no idéntico en la gran mayoría de ocasiones.

Preparación de los datos

Se realizan **5 iteraciones** del algoritmo para un mismo *dataset* en cada ejecución del archivo. **Para cada iteración se toman distintos valores para los sets de entrenamiento y pruebas**, utilizando el **%80** de las muestras para **entrenamiento** y **%20** para **pruebas**. Véase en la siguiente fracción de código:

```
for i in range(5):
    print('\nIteration: ',i+1, '\n')
    x_train = []; y_train = []; x_test = []; y_test = []

    # fill the train/test data sets with random samples
    for j in range(len(samples)):
        n = random.randint(0,m-1)
        # Training: %80 of samples
        if(j <= len(samples)*.8 ):
            #train data set
            x_train.append(samples[n])
            y_train.append(y[n])
        # Pruebas: %20 of samples
        else:
            x_test.append(samples[n])
            y_test.append(y[n])
```

Rubén Sánchez Mayén A01378379

Análisis de precisión

El número de muestras **(m)** y el tamaño de paso **(*alpha*)** están correlacionadas negativamente para la implementación de este algoritmo. A medida que **se incrementa el número de muestras**, **la alfa puede quedar obsoleta si no se disminuye**, causando **errores** en los cálculos de los coeficientes propuestos.

Para:

```
m = 10
alpha = .01
```

Se obtiene:

```
final params: [2.6988544300535824, 5.767260468248669, 3.068406038195087, 0.3695516081415076]
Average square error: 2.5045096101200515
Estimated Y for [1, 4, 2, 8] : 34.86112124457049
```

Para:

```
m = 50
alpha = .01
```

Se obtiene:

```
final params: [nan, nan, nan, nan]
Average square error: nan
Estimated Y for [1, 4, 2, 8] : nan
```

Nótese que en ambas en ambas implementaciones se utiliza el tamaño de paso ***(alpha)*** de **.01**, pero se incrementa de **10 muestras** a **50 muestras.** En la segunda ejecución hubo errores en los cálculos.

Para:

```
m = 50
alpha = .0001
```

Se obtiene:

```
final params: [0.8266065493570572, 4.046385024499528, 3.219778475142468, 2.3931719257854183]
Average square error: 2.2186943014535365
Estimated Y for [1, 4, 2, 8] : 42.59707900392345
```

Nótese cómo ahora que el ***alpha*** se disminuyó a **.0001** y se mantuvo **el mismo número de muestras de 50**. En esta ocasión los resultados son calculados exitosamente y **el error** disminuye en comparación de tener **10 muestras**, no obstante, esto podría atribuirse cierta medida a la **aleatoriedad** en la generación de las **muestras**, ya que estas cambian con cada ejecución del archivo. Es una mejor opción únicamente calcular las muestras una vez, y mantener los datos en un ***archivo csv*** o similar.

A pesar de la **aleatoriedad**, la mejora del modelo según su número de muestras es visible.

```
 56   m = 100
 57   alpha = .0001

final params: [0.22138342441162592, 3.4250780566421852, 3.2036946322305573, 2.982311207818931]
Average square error: 0.3300186643259632
Estimated Y for [1, 4, 2, 8] : 44.18757457799293
```