Regresión Lineal

Somos el CEO de una empresa de desarrollo de videojuegos con sede en España y queremos abrir una nueva sucursal de desarrollo de nuestra empresa en otro país.

Para decidir donde abrirla, el equipo económico ha recopilado datos de renta media de diferentes países y el beneficio estimado de realizar el siguiente videojuego en cada país, en base al beneficios obtenidos por otras empresas de la competencia al realizar juegos similares. Los beneficios estimados no sólo dependen del coste de los trabajadores, dependen tambien del coste de la sede, impuestos, etc. Pero desde el departamento económico, creen que éste es el factor más influyente y que otros factores (cómo el precio del local) están correlacionados con la renta media. Además, piensan que la distribución de los datos sigue un progresión lineal, es decir, que cuanto mayor sea la renta media del país, más beneficios se podrán obtener en el videojuego. Esto parece contraintuitivo, pero podemos ver ejemplos de juegos con muchos beneficios creados en países con rentas altas como Japón, EEUU o Reino Unido.

Se pide crear un modelo que prediga, dado los datos de un país cualquiera, cual sería el beneficio esperado del videojuego si se desarrollase en dicho país.

Para ello se proporciona un dataset con los datos del salario medio anual en miles de Euros de cada uno de los países y el beneficio medio obtenido por las empresas del sector que crearon juegos similares en dicho país, en millones de euros.

Como ejemplo si vemos el primer caso del dataset (6.1101,17.592) nos indica que el país 1 tiene una renta media de 6110,7 euros y se obtuvieron 17,592 millones de euros de beneficio medio.

Ficheros incluidos en la práctica: - data/ex1data1.txt dataset con los datos de partida. - public_tests.py: funciones para comprobar que el ejercicio es correcto - utils.py: funciones para cargar los datos. - linear_reg.py: Fichero donde tendréis que implementar la regresión lineal

Se pide:

1. Visualiza los datos del dataset para corroborar visualmente que se ajustan a una recta
2. Calcula la función de coste $J(w, b)$ (véase apuntes del Tema 2, sección 1). La función que tendréis que implementar es la siguiente:

```
#Args:
#x (ndarray): Shape (m,) Input to the model
#y (ndarray): Shape (m,) Estimate profits
#w, b (scalar): Parameters of the model
```

```
def compute_cost(x, y, w, b):
    return total_cost
```

3. Calcula el gradiente (véase apuntes del Tema 2, sección 1)

```
#Args:
#x (ndarray): Shape (m,) Input to the model
#y (ndarray): Shape (m,) Estimate profits
#w, b (scalar): Parameters of the model
#Return: gradient of each parameters.

def compute_gradient(x, y, w, b):
    return dj_dw, dj_db
```

4. Descenso de gradiente

```
#Args:
#x (ndarray): Shape (m,) Input to the model
#y (ndarray): Shape (m,) Estimate profits
#w_in, b_in (scalar): Initial parameters of the model
#cost_function: function to compute cost
#gradient_function: function to compute the gradient
#alpha : (float) Learning rate
#num_iters : (int) number of iterations to run gradient descent
#Return:
# w, b: Updated values of parameters of the model after running gradient descent
# J_history: (ndarray): Shape (num_iters,)
# J at each iteration primarily for graphing later

def gradient_descent(x, y, w_in, b_in, cost_function, gradient_function,
alpha, num_iters):
    return w, b, J_history
```

5. Dibuja la curva de aprendizaje usando la historia de la función gradient_descent

6. Dibuja la recta de regresión usando el modelo optimizado por descenso de gradiente.

Entregar un zip con: - Los ficheros de la práctica en Python. - Un Jupiter Notebook con la ejecución de los ejercicios y el dibujado de las gráficas.

Nota: Para usar las funcione en Jupiter Notebook copiad los ficheros .py en la raíz junto al notebook e importarlos de la siguiente forma:

```
from utils import load_data
from linear_reg import gradient_descent
```

```
from linear_reg import compute_gradient
from linear_reg import compute_cost
```

English version:

We are the CEO of a video game development company with headquarters in Spain and we want to open a new development branch of our company in another country.

To decide where to open it, the economic team has collected data on the average income of different countries and the estimated profit of making the next videogame in each country, based on the profits obtained by other competing companies when making similar games.

The estimated profits do not only depend on the cost of the workers, but also on the cost of the headquarters, taxes, etc. But from the economic department, they believe that this is the most influential factor and that other factors (such as the price of the headquarter) are correlated with the average income. Moreover, they believe that the distribution of the data has a linear progression, i.e. the higher the average income of the country, the more profits can be made in the video game. This seems counter-intuitive, but we can see examples of games with high profits created in high-income countries such as Japan, the US or the UK.

We are requested to create a model that predicts, given the data of any country, the expected profit of the video game would be if it were developed in that country.

To do this, a dataset is provided with data on the average annual income (in thousands of euros) for each of the countries and the average profit obtained by the companies in the sector that created similar games in that country (in millions of euros)

As an example, if we look at the first case of the dataset (6.1101,17.592) it indicates that country 1 has an average income of 6110.7 euros and 17.592 million euros of average profit were obtained.

Files included in the practice: - data/ex1data1.txt dataset with the starting data. - public_tests.py: functions to check that the exercise is correct. - utils.py: functions to load the data. - linear_reg.py: file where you will have to implement the linear regression.

You are asked for: 1. Display the data in the dataset to visually corroborate that it fit to a line. 2. Calculate the cost function $J(w, b)$ (see notes in Topic 2, section 1). The function you will need to implement is as follows:

```
#Args:
#x (ndarray): Shape (m,) Input to the model
```

```
#y (ndarray): Shape (m,) Estimate profits
#w, b (scalar): Parameters of the model

def compute_cost(x, y, w, b):
    return total_cost
```

3. Calculate the gradient (see notes in Topic 2, section 1).

```
#Args:
#x (ndarray): Shape (m,) Input to the model
#y (ndarray): Shape (m,) Estimate profits
#w, b (scalar): Parameters of the model
#Return: gradient of each parameters.

def compute_gradient(x, y, w, b):
    return dj_dw, dj_db
```

4. Gradient descent

```
#Args:
#x (ndarray): Shape (m,) Input to the model
#y (ndarray): Shape (m,) Estimate profits
#w_in, b_in (scalar): Initial parameters of the model
#cost_function: function to compute cost
#gradient_function: function to compute the gradient
#alpha : (float) Learning rate
#num_iters : (int) number of iterations to run gradient descent
#Return:
# w, b: Updated values of parameters of the model after running gradient descent
# J_history: (ndarray): Shape (num_iters,)
# J at each iteration primarily for graphing later

def gradient_descent(x, y, w_in, b_in, cost_function, gradient_function,
alpha, num_iters):
    return w, b, J_history
```

5. Draw the learning curve using the history of the gradient_descent function.
6. Draw the regression line using the optimised gradient descent model.

Deliver a zip file with: - The Python files. - A Jupiter Notebook with the execution of the exercises and the drawing of the graphs.

Note: To use the functions in Jupiter Notebook copy the .py files in the root close to the notebook and import them:

```
from utils import load_data
from linear_reg import gradient_descent
```

```
from linear_reg import compute_gradient
from linear_reg import compute_cost
```