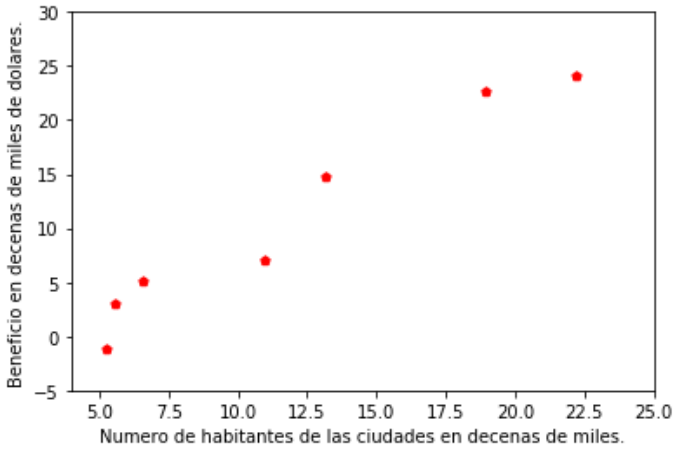
REGRESIÓN LINEAL

Solución directa

Como primera aproximación para la regresión vamos a implementar la solución análitica o directa. Recordemos que tenemos que estimar los parámetros θ, que en caso de la regresión con una única variable son $\theta 0$ y $\theta 1$ tal que $y=\theta 0+\theta 1 \cdot x$ .Para ello, como ya tenemos los datos en la matriz X con la primera columna siendo todo unos, simplemente obtenemos los valores de los parámetros aplicando la siguiente ecuación.

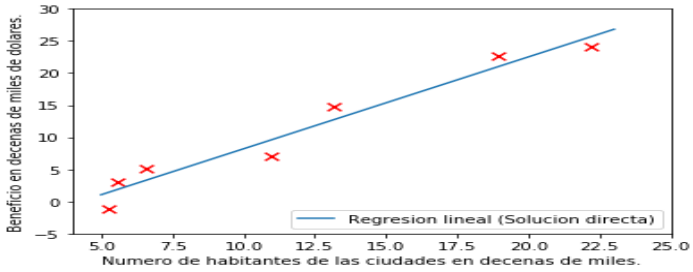
$$\theta=(X^T X)^{-1} X^T y$$

```
Datos de entrada X
[[ 1.    6.56]
 [ 1.    5.56]
 [ 1.   18.94]
 [ 1.   10.95]
 [ 1.   13.17]
 [ 1.   22.2 ]
 [ 1.    5.25]]
Datos de salida y
[[ 5.18]
 [ 3.08]
 [22.63]
 [ 7.04]
 [14.69]
 [24.14]
 [-1.22]]
```



Ahora vamos a mostrar la recta aprendida. Para ello debéis crear una array de valores comprendidos entre 5 y 23 con incrementos de 1 en 1 y almacenarlo en la variable xx. A continuación evaluar todos estos valores utilizando los valores del modelo aprendidos anteriormente (variable theta) y almacenarlos en la variable yy. Se debe aplicar la ecuación de la recta: $y=\theta_0+\theta_1 \cdot$

```
theta
[[-6.11039169]
 [ 1.43183761]]
Array xx
[ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
Array yy
[ 1.04879636  2.48063398  3.91247159  5.3443092   6.77614681  8.20798442
  9.63982204 11.07165965 12.50349726 13.93533487 15.36717248 16.79901009
 18.23084771 19.66268532 21.09452293 22.52636054 23.95819815 25.39003576
 26.82187338]
```



Descenso por gradiente

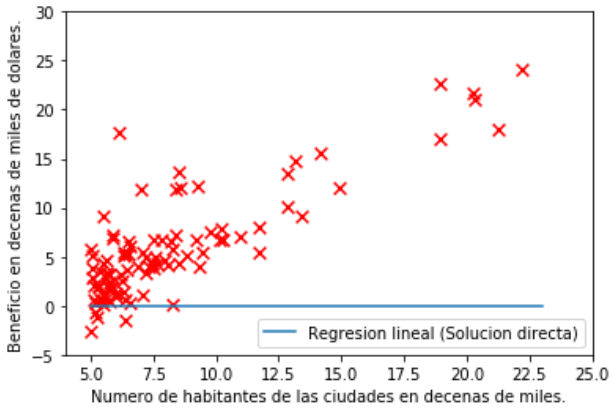
Como ya hemos visto en teoría, el método directo tiene ciertos inconvenientes. En esos casos, el método a utilizar es el descenso por gradiente. Ahora vamos a seguir una serie de pasos para crear nuestro algoritmo de aprendizaje completo.

En primer lugar vamos programar la función de coste para poder aplicar el algoritmo de aprendizaje y que los parámetros de la regresión lineal se ajusten a los datos de tal forma que se minimice dicho coste. Recordar que la función de coste es:

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

-m: nº de muestras del conjunto de dato
-h (θ): multiplicación fila X (muestra i) por theta
-y: Valor de salida(muestra i)

Ejemplo con valores theta θ_0=0 y θ_1=0



La función de coste mide la suma de las distancias de cada muestra a la recta, en este caso la recta esta en el eje x, por lo tanto la función de coste es alta (`[32.07273388]`)

A continuación vamos a implementar la función del descenso por gradiente para realizar el aprendizaje de los parámetros del modelo. Esta función recibe como parámetros de entrada los valores de entrada, de salida ,los valores iniciales del modelo (theta), el valor que establece la velocidad de convergencia del algoritmo (α) y el número de iteraciones a realizar por el algoritmo.

Como salida devuelve los parámetros aprendidos (theta), así como el historial de la evolución de error cometido con cada configuración de valores del modelo (variable J_history). Fijaros que esta última variable almacena el resultado de la función de coste implementada anteriormente para los valores del modelo que han sido actualizados en la instrucción anterior.

En cada iteración se deben aplicar los siguientes pasos:

1. Calcular la salida del modelo $h\theta(x(i))$ para todo $i \in 1,...,m.$ y calcular la función de coste como se muestra anteriormente.
2. Actualizar los valores de θ aplicando:

$$\theta = \theta - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Ten en cuenta que puedes actualizar todos los parámetros de manera simultánea mediante operaciones entre matrices. Tienes los valores de salida hθ(x) en un vector(mx1).Calcula la diferencia con y(mx1). Ahora si haces la multiplicación matricial entre X^T(2xm) y el vector diferencia(mx1)obtendrás el resultado correspondiente al sumatorio. Es decir, la fórmula queda de la siguiente forma:

$$\theta = \theta - \alpha \cdot \frac{1}{m} X^T \cdot (h_\theta(x) - y)$$

En este caso el resultado es un vector columna de dos valores (theta).

Ejemplo de código

```
def gradientDescent(X, y, theta, alpha=0.01, num_iters=1500):
    m = y.size
    J_history = np.zeros(num_iters)
    for iter in np.arange(num_iters):
        h=X.dot(theta)
        theta = theta-((alpha/m)*np.dot(X.T,h-y))
        coste=computeCost(X,y,theta)
        J_history[iter] =coste
    return(theta, J_history)
```

SCIKIT-LEARN

Una vez realizado el proceso de aprendizaje con funciones definidas por nosotros vamos a pasar a usar la librería ofrecida por Python para realizar tareas de aprendizaje automático. Esta librería se llama scikit-learn.

Hoy vamos a comenzar utilizando las funcionalidades relacionadas con la regresión lineal:

- Para ello hay que importar la clase LinearRegression del paquete linear_model ofrecido en scikit-learn (la librería se llama sklearn).
- Una vez importada la clase debemos llamar al constructor para inicializar el modelo de regresión lineal.
- Después hay que realizar el aprendizaje y para ello se utiliza la función fit, a la que se le pasan los datos de entrada (X) y las salidas correspondientes (y). Esta función automáticamente realiza el aprendizaje y modifica internamente los parámetros del modelo intercept_ y coef_ para el término independiente y los θ asociados a cada variable del problema.
- Para clasificar nuevos datos tenemos dos opciones
    - Utilizar intercept_ y coef_ aplicando la ecuación de la regresión lineal: $y=intercept\_+coef\_\cdot x$
    - Utilizar la función predict, a la que se le pasa los datos de entrada de las instancias a predecir y devuelve las predicciones.

En el código mostrado a continuación se ve un ejemplo del uso de esta librería con comentarios sobre qué se realiza en cada instrucción de código. Finalmente se muestra en una gráfica tanto la recta aprendida anteriormente como la recta aprendida con la librería scikit-learn.

```
# Se importa de la libreria scikit-learn de python (sklearn) el paquete correspondiente a la regresión lineal
from sklearn.linear_model import LinearRegression

# Se incializa el modelo llamando al constructor de la regresión lineal: todos los parámetros se asignan a sus valores por defec
regr = LinearRegression()
# Se entrena el modelo (aprendizaje) utilizando la variable de entrada sin los unos X[:,1] y la variable de salida (y)
# La función ravel devuelve una lista (si la entrada es una lista de listas coge todos los elementos y los almacena en la lista
# En la función reshape el valor -1 significa que el tamaño de esa dimensión se calcula automáticamente en base a la otra
    # En este caso hacemos una columna de tantas filas como instancias tengan los datos
regr.fit(X[:,1].reshape(-1,1), y.ravel())
# Se evalúan los valores creados anteriormente para comparar ambos modelos aprendidos
prediccionesOpcion1 = regr.intercept_+regr.coef_*xx
prediccionesOpcion2 = regr.predict(xx.reshape(-1,1))
plt.plot(xx, prediccionesOpcion2, label='Regresion lineal (Scikit-learn)')
# Se muestra los datos a partir de los cuales se ha aprendido
plt.scatter(X[:,1], y, s=30, c='r', marker='x', linewidths=1)
# Se muestra la recta aprendida anteriormente para comparar
plt.plot(xx,yy, label='Regresion lineal (Descenso por gradiente)')

plt.xlim(4,24)
plt.xlabel('Numero de habitantes de las ciudades en decenas de miles.')
plt.ylabel('Beneficio en decenas de miles de dolares.');
plt.legend(loc=4);
```

Predicción de nuevos datos

Una vez aprendido el modelo de regresión lineal podemos utilizarlo para predecir datos que no han sido utilizados durante el entrenamiento. En este caso debéis predecir el valor de dos ciudades: la primera tiene 35.000 habitantes y la segunda 70.000 habitantes.

```
# Predecir el beneficio para dos ciudades de 35000 y 70000 habitantes, respectivamente
#ciudad1 = regr.intercept_+regr.coef_*35000#<RELLENAR>
ciudad1= theta[0,0]*10000.0+theta[1,0]*35000.0
#ciudad2 = regr.intercept_+regr.coef_*70000#<RELLENAR>
ciudad2= theta[0,0]*10000.0+theta[1,0]*70000.0
print(ciudad1)
Test.assertEquals(round(ciudad1, 5), 4519.76787, 'Valor de la ciudad 1 incorrecto')
print(ciudad2)
Test.assertEquals(round(ciudad2, 5), 45342.45013, 'Valor de la ciudad 2 incorrecto')
```

```
4519.767867701768
1 test passed.
45342.45012944714
1 test passed.
```