1RegresionLineal1Variable

June 19, 2019

```
In [ ]: #José Antonio Garrido Sualdea
```

```
In [24]: ############################
         #REGRESION LINEAL 1 VARIABLE
         ############################
```

```
In [25]: import pandas as pd #perimte trabajar con los datos de forma tabular
         import matplotlib.pyplot as plt
```

```
In [26]:  #dataframe donde cargamos los datos
         nombres_columnas = ['poblacion', 'beneficio']
         data = pd.read_csv('ex1data1.csv', names=nombres_columnas)
         data.head()
```

Out[26]:

	poblacion	beneficio
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

```
In [27]: #hipotesis, predicción del beneficio en función de la poblacion
         def h(x, t): #t = teta
             return t[0] + t[1]*x

         def J(data, t): #función de coste
             #columna nueva 'error cuadrático'
             data['error cuadratico'] = data.apply(
                 lambda row: #funcián sin nombre, row es el parámetro
                     (
                         h(row['poblacion'], t) #predicción
                          - row['beneficio'] #valor real
                     )**2,
                 axis=1 #aplica la función fila a fila
             )
             return (
                 1/(2*data['poblacion'].count())
             ) * data['error cuadratico'].sum()
```

```
def gradiente(data, t, j, alfa=0.01):
    #columna nueva 'gradiente'
    data['gradiente'] = data.apply(
        lambda row:
            (#cuanto se equivoca
                h(row['poblacion'], t)
                - row['beneficio']
            )
            #utiliza el valor de la columna para que
            #el ajuste sea de la misma escala(tamaño)
            #j==0 es la constante, por eso tiene siempre valor 1
            * (row['poblacion'] if j==1 else 1)
        , axis=1
    )
    return (
        t[j] - (#media de los gradientes y lo controla con alfa
            alfa * data['gradiente'].sum() / data['poblacion'].count()
        )
    )
```

```
In [28]: def plot_data(data, t): #función pintar gráfica
             fig = plt.figure()
             data.plot.scatter(x='poblacion', y='beneficio')
             min_value = data['poblacion'].min()
             max_value = data['poblacion'].max()
             plt.plot([min_value, max_value],[h(min_value, t), h(max_value, t)])
             plt.show()
```

```
In [29]: t = [1, 1]
         print('configuracion inicial - t: {}, J: {}'.format(t, J(data, t)))
         #gradiente(data, t, 0)
         #gradiente(data, t, 1)
         #data.head()
```

```
configuracion inicial - t: [1, 1], J: 10.266520491383504
```

```
In [30]: MAX_ITERACIONES = 1500
         coste_x_iteracion = []
         for i in range(0, MAX_ITERACIONES):
             for j in range(len(nombres_columnas)):
                 t[j] = gradiente(data, t, j)
                 #vamos guardando el coste(j) por iteración
                 #para ver la evolución del apredizaje
                 coste_x_iteracion.append(J(data, t))

             if i % 500 == 0:
                 print('iter: {}, t: {}, J: {}'.format(i, t, J(data, t)))
```

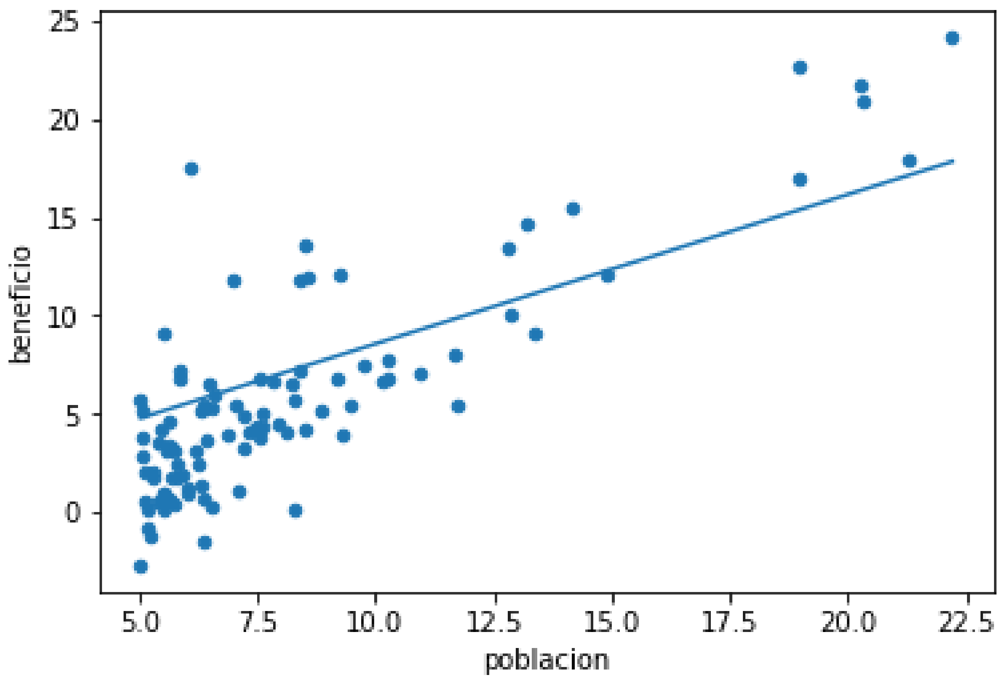
```
        plot_data(data, t)

print('evolucion del coste segun el numero de iteracion')
#mostramos la gráfica de la evolución del aprendizaje
fig = plt.figure()
plt.plot(range(len(coste_x_iteracion)), coste_x_iteracion)

data.head()
```

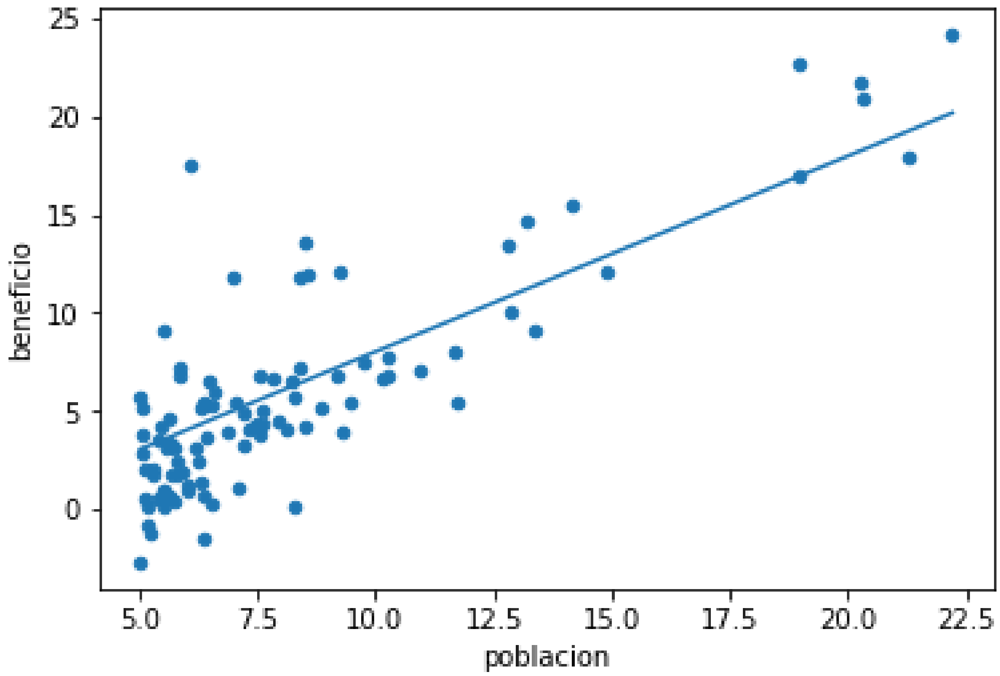
```
iter: 0, t: [0.9667933505154639, 0.7603606654195877], J: 6.75149515419807
```

```
<Figure size 432x288 with 0 Axes>
```



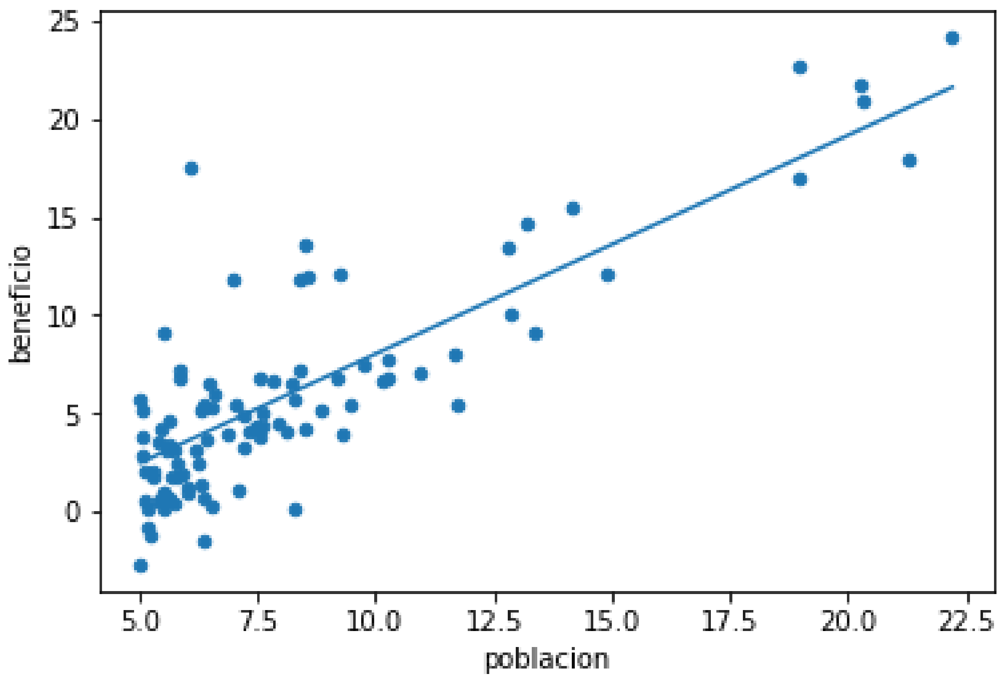

```
iter: 500, t: [-1.9397151324962003, 0.9968791839002915], J: 4.825298331961877
```

```
<Figure size 432x288 with 0 Axes>
```




```
iter: 1000, t: [-3.108022511435824, 1.1140371597291412], J: 4.533465884896614
```

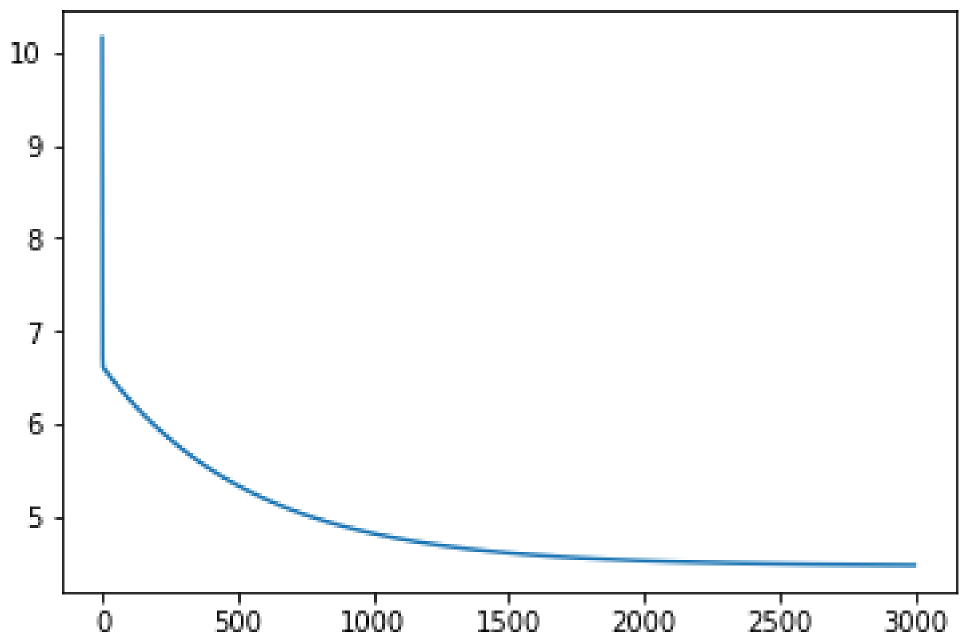
```
<Figure size 432x288 with 0 Axes>
```



evolucion del coste segun el numero de iteracion

Out[30]:

	poblacion	beneficio	error cuadratico	gradiente
0	6.1101	17.5920	198.109507	-86.002667
1	5.5277	9.1302	39.559050	-34.768787
2	8.5186	13.6620	54.000154	-62.602970
3	7.0032	11.8540	53.291523	-51.126936
4	5.8598	6.8233	12.938961	-21.080143



In [31]:
```
#####################################
#REGRESION LINEAL MULTIPLES VARIABLES
#####################################
```

In [32]:
```
import pandas as pd #perimte trabajar con los datos de forma tabular
import matplotlib.pyplot as plt
import numpy as np
```

In [33]:
```
#tamaño en pies cuadrados, el número de habitaciones y el precio
#dataframe donde cargamos los datos
nombres_columnas = ['piesCuadrados', 'nHabitaciones', 'precio']
data = pd.read_csv('ex1data2.csv', names=nombres_columnas)
data.head()
```

Out[33]:

	piesCuadrados	nHabitaciones	precio
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

In [34]:
```
X = data[['piesCuadrados', 'nHabitaciones']]
y = data ['precio']
```

In [35]:
```
def normalizar(fila, mu, sigma):
    return (fila - mu) / sigma
```

In [36]:
```
def J(X_norm, T):
    return (X_norm.dot(T) - y).transpose().dot(X_norm.dot(T) - y) / (2*m)
```

In [37]:
```
def gradiente(X_norm, T, alfa):
    coste_x_iteracion = []
    MAX_ITERACIONES = 1000
    for i in range(MAX_ITERACIONES):
        for t in range(n):
            T[t] = T[t] - (alfa/m)*((X_norm.dot(T) - y)
                   .transpose()*X_norm[X_norm.columns[t]]).sum()

        coste_x_iteracion.append(J(X_norm, T))

    return coste_x_iteracion
```

In [38]:
```
mu = X.mean() #media vectorizada
sigma = X.std() #desviación vectorizado

#axis=1 aplica la función para cada fila
X_norm = X.apply(lambda fila: normalizar(fila, mu, sigma), axis=1)
X_norm['constante'] = 1
(m, n) = X_norm.shape
X_norm.head()
```

Out[38]:

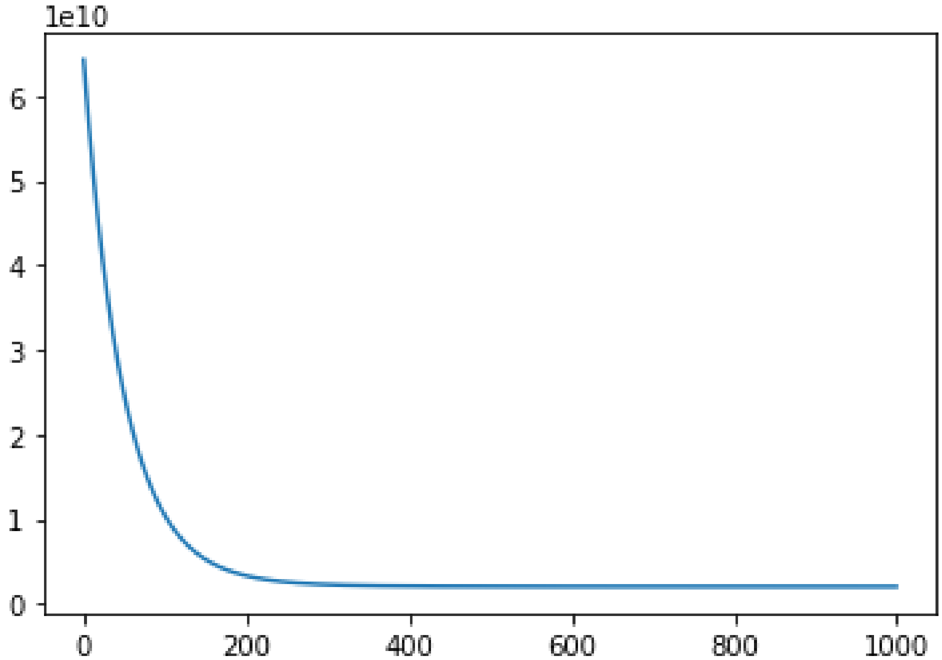
	piesCuadrados	nHabitaciones	constante
0	0.130010	-0.223675	1
1	-0.504190	-0.223675	1
2	0.502476	-0.223675	1
3	-0.735723	-1.537767	1
4	1.257476	1.090417	1

In [39]:
```
T = np.array([1]*n)
coste_x_iteracion = gradiente(X_norm, T, 0.01)
print('evolucion del coste segun el numero de iteracion')
```

```
evolucion del coste segun el numero de iteracion
```

In [40]:
```
#mostramos la grafica de la evolución del aprendizaje
fig = plt.figure()
plt.plot(range(len(coste_x_iteracion)), coste_x_iteracion)
```

Out[40]: [<matplotlib.lines.Line2D at 0x16f3cdf8400>]



In [41]:
```
#normalizo y preparo los datos de prueba
prueba = np.array([1650, 3])
prueba_norm = normalizar(prueba, mu, sigma)
prueba_norm['constante'] = 1
print(prueba_norm)
```

```
piesCuadrados   -0.441273
nHabitaciones   -0.223675
constante        1.000000
dtype: float64
```

In [42]:
```
#hipótesis o predicción segun lo aprendido
#en el proceso iterativo
prueba_norm.dot(T)
```

Out[42]: 293172.9955954961

In [43]:
```
#ecuación nomral
T_normal = ((np.linalg.inv((X_norm.transpose().dot(X_norm))))
            .dot(X_norm.transpose())).dot(y)
```

In [44]:
```
#theta para el proceso iterativo
print (T)
#y theta para la ecuación normal
print(T_normal)
```

```
[109753  -5772 340313]
[110631.05027885  -6649.47427082 340412.65957447]
```

In [45]:
```
#hipotesis o predicción segun la ecuación normal
prueba_norm.dot(T_normal)
```

Out[45]: 293081.46433489607