GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE HUELVA

Explicación de la práctica Regresión Lineal y la elección del alfa

Memoria de la actividad individual

Autor: Wadadi sidelgaum limam

Contenido

1. Introducción

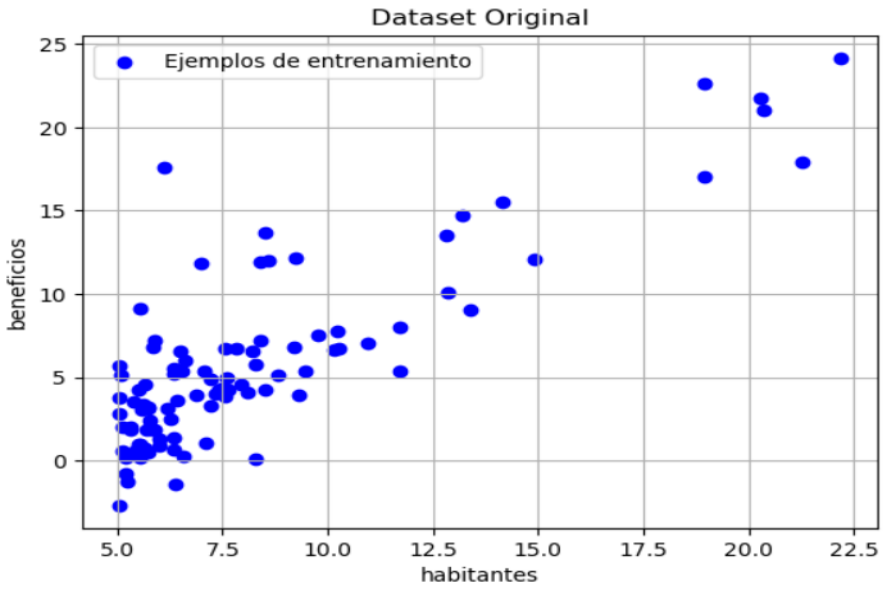
La práctica consiste en implementar los métodos de regresión lineal visto en clase, usando el contenido del fichero (regresión_1.csv).

Los métodos que usamos para el cálculo de los coeficientes de la regresión lineal son:

- **Aplicación Externa**: Utilizando la aplicación Excel calculamos los coeficientes de la recta de regresión lineal para tener una primera aproximación.
- **Ecuaciones normales (matrices)**: Utilizamos el cálculo de matrices mediante la librería numpy de Python para obtener los valores de ajuste de la regresión lineal.
- **Descenso por Gradiente**: Obtenemos los coeficientes mediante la implementación del algoritmo descenso por gradiente en Python.
- **Descenso por Gradiente Estocástico**: Mediante la implementación del algoritmo descenso por gradiente Estocástico en Python obtenemos los coeficientes de la regresión lineal.

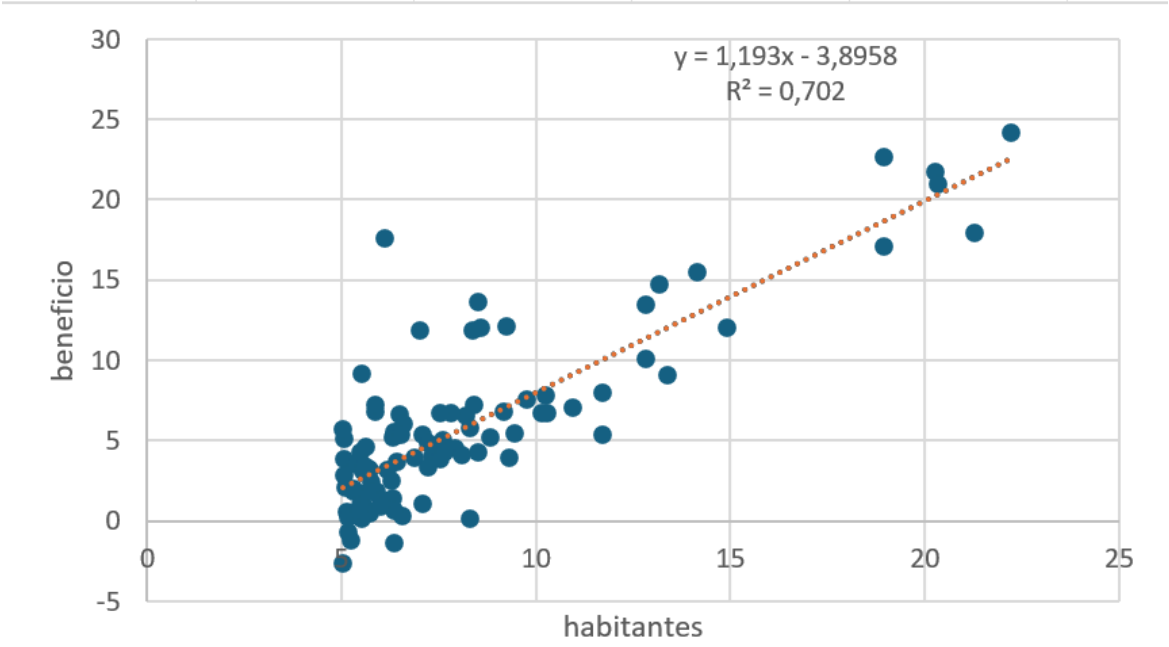
2. Representación gráfica de los datos:

Para representar gráficamente los ejemplos hemos hecho uso de la librería matplotlib de Python.



3. Aplicación Externa: Excel

Mediante la aplicación podemos obtener los valores de los coeficientes de regresión lineal y la representación gráfica de los datos y la línea de tendencia.



4. Ecuaciones normales (matrices):

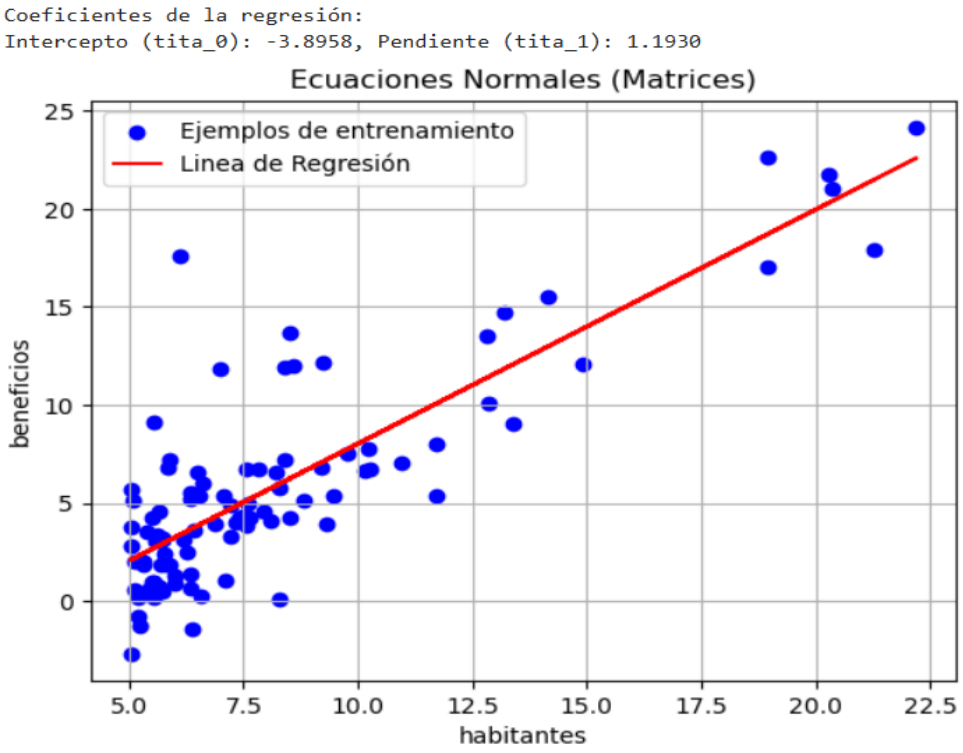
En ecuaciones normales calculamos los coeficientes aplicando la definición analítica de tita, la formula matricial de las ecuaciones normales para obtener las distintas titas es la siguiente:

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

A continuación, vemos el desarrollo de la formula anterior en lenguaje Python:

```
# leemos los datos del fichero
datos=panda.read_csv("regresion_1.csv",header=None)
# extraemos los valores de atributos (habitantes) y los objeticos (beneficio)
x=datos.iloc[:,0]
y= datos.iloc[:,1]
# obtener los valores de entrada y salida en formato columna para hacer el calculo de la ecuacion normal
X = np.matrix(x).T
y = np.matrix(y).T
# añadir una columna de unos al matriz X para el termino independiente (tita_0)
X_b = np.hstack([np.ones((X.shape[0], 1)), X])
tita = np.linalg.inv(X_b.T * X_b) * (X_b.T * y)
```

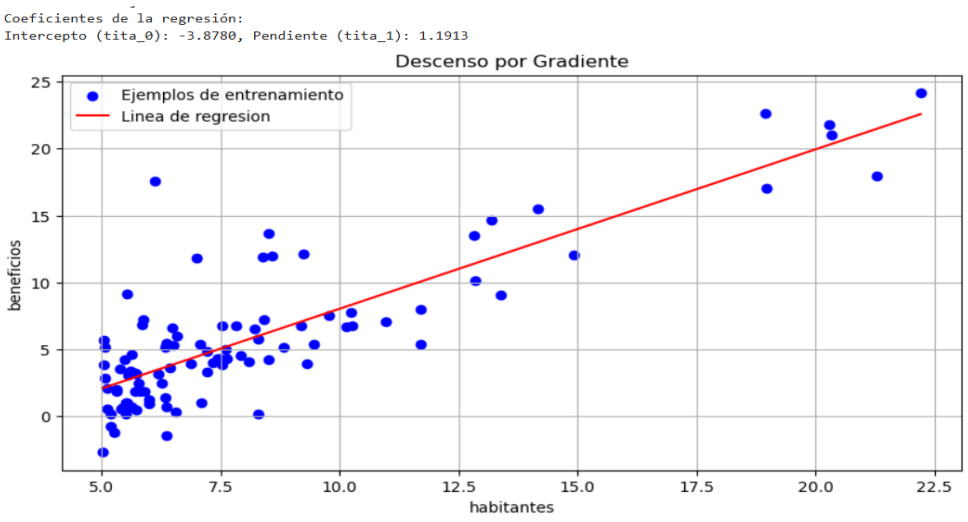
La representación gráfica de los datos con la línea de regresión es la siguiente:



5. Descenso por gradiente:

Este algoritmo se trata de un algoritmo iterativo que consiste en encontrar los valores de los coeficientes de la regresión mediante la minimización del error.

En la siguiente grafica observamos los valores de los coeficientes con la recta de la regresión:



En la gráfica anterior, vemos los resultados de la regresión, calculados a partir de una ratio de aprendizaje equivalente a 0.0001 escogido a mano entre 5 distintos valores y un criterio de convergencia de 3000 iteraciones.

Para elegir una buena alfa hay que tener en cuenta que:

- Si es demasiado grande, los cambios en la pendiente serán también muy grandes y será difícil encontrar los coeficientes que minimicen la función de coste, es lo que ocurre en el caso del que el alfa valga 0,01.
- es demasiado pequeño, el gradiente descendiente tardará mucho en encontrar la solución adecuada, ya que se necesita más pasos hasta alcanzar el mínimo, es lo que ocurre cuando el alfa es igual a 0,000001.

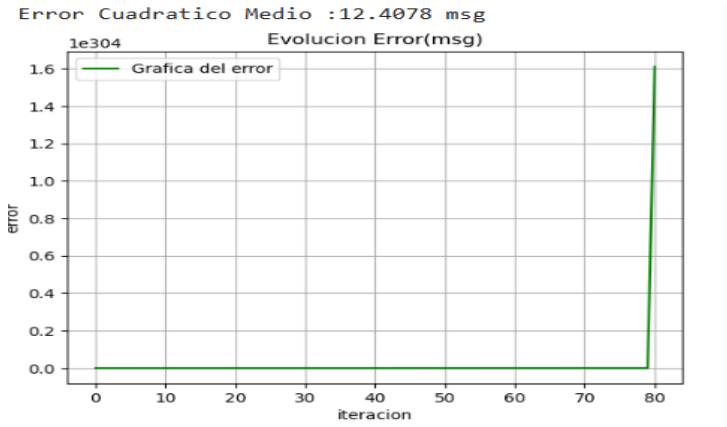
El valor 0.0001, proporciona un ajuste mejor, ya que se minimiza el error cuadrático medio más rápidamente en comparación con valores más bajos.

Una vez escogido la ratio de aprendizaje, para el criterio de convergencia hemos seleccionado un valor fijo de 3000 iteraciones, que fue ajustado en función del comportamiento del algoritmo, asegurando que tuviera tiempo suficiente para encontrar los coeficientes óptimos sin realizar iteraciones necesarias.

Por otro lado, en las siguientes graficas observamos la evolución del error con distintos valores de la tasa de aprendizaje, para ello, se ha hecho el uso de la siguiente fórmula para el cálculo del error total cuadrático.

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

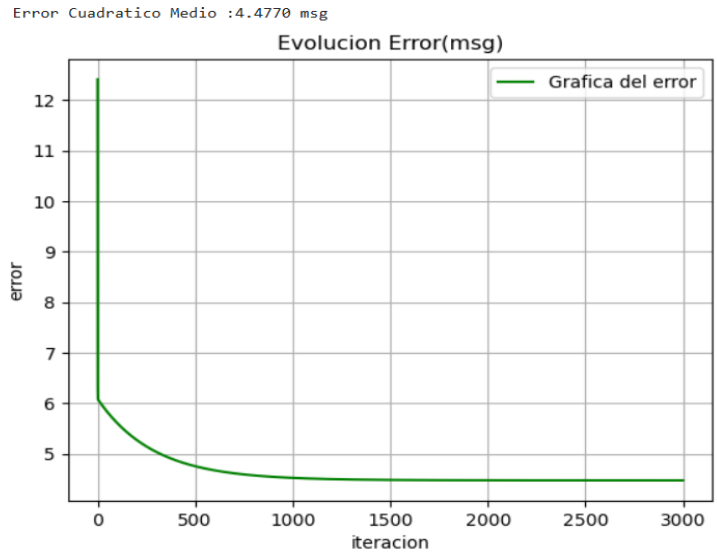
1. Grafica del error con un valor de alfa equivalente a 0.01:



Como observamos en la figura anterior, parece que el error crece de manera exponencial después de 78 iteraciones, ya que el modelo realiza grandes

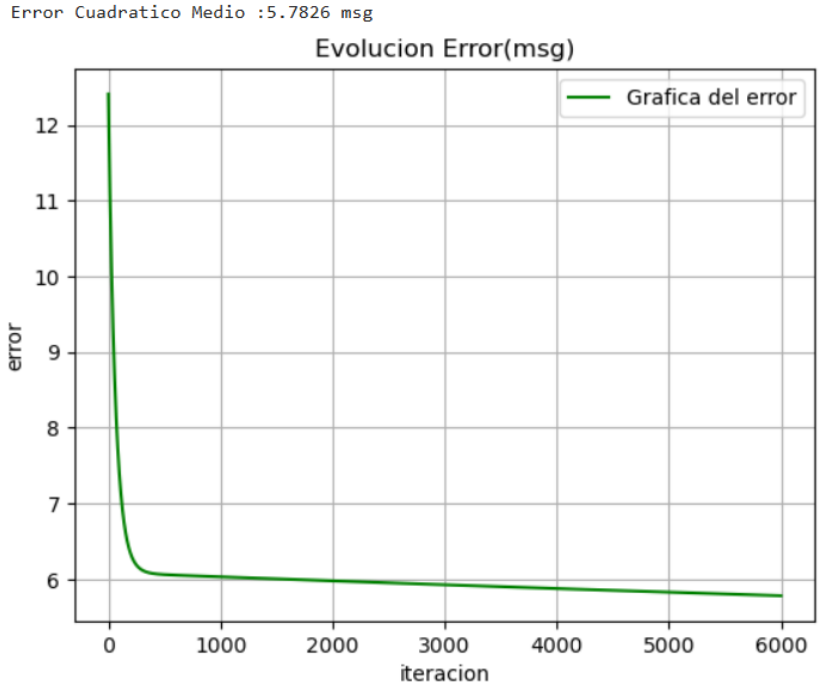
saltos en cada modificación de las titas, al ser el valor del ratio demasiado alto.

2. Grafica del error con un valor de alfa equivalente a 0.0001:



En la gráfica se observa que con un valor de alfa equivalente a 0,0001, hace que el error decrece constantemente y de manera progresiva, hasta llegar a estabilizarse en un valor mínimo sin grandes fluctuaciones después de la iteración 1000.

3. Grafica del error con un valor de alfa equivalente a 0.000001:



En esta gráfica, se muestra una disminución inicial del error significativa, pero después de las primeras 400 iteraciones, el descenso se desacelera suavemente y el error no baja tanto comparado con la gráfica del 0,0001, finalmente, queda el error estabilizándose en un valor más alto.

Como hemos visto en los resultados de las gráficas anteriores, la evolución del error cuadrático medio según el número de iteraciones del criterio de parada, también observamos que no todos los valores de la tasa de aprendizaje proporcionan buena aproximación, por lo que hemos escogido el segundo ajuste (valor de alfa 0,0001) ya que es el que obtiene un ajuste mejor de los tres.

A continuación, adjuntamos el código que desarrolla el algoritmo:

```
alfa = 0.0001 # inicializamos la tasa de aprendizaje
conj=[123456789,234567890,345678900,456789000,567890000] # conjuntos que contiene las semillas que usaremos
resultados_error = []
resultados_tita_0 = []
resultados_tita_1 = []
for k in conj:
 # diferentes semillas
 random.seed(k)
 # incializamos las titas aa valores aleatorios entre 0 y 1
 tita_0 = random.random()
 tita_1 = random.random()
 conj_errores = []
 conj_tita_0 = []
 conj_tita_1 = []
 for i in range(3000): # numero de iteraciones para el criterio de parada (30000 iteraciones hasta que converga)
   sum_0, sum_1,error  = 0, 0, 0
   for j in range(len(x)): # calculamos el sumatorio del gradiente y lo multiplicamos por el alpha
     h_x = tita_0 + tita_1 * x.iloc[j]
     sum_0 += (h_x - y.iloc[j])
     sum_1 += ((h_x - y.iloc[j]) * x.iloc[j])
     error += (h_x - y.iloc[j]) ** 2
   # modificamos los coeficientes restando el valor obtenido para que decrececa las titas en sentido contrario
   tita_0 -= alfa * sum_0
   tita_1 -= alfa * sum_1
   # añadimos los datos(valores de los coeficientes y el error) a los conjuntos
   conj_errores.append(0.5 * error / len(x))
   conj_tita_0.append(tita_0)
   conj_tita_1.append(tita_1)
 # calculamos el valor que minimiza el error para obtener los coeficientes que le corresponden
 mini = min(conj_errores)
 ind = conj_errores.index(mini)
 resultados_error.append(mini)
 resultados_tita_0.append(conj_tita_0[ind])
```

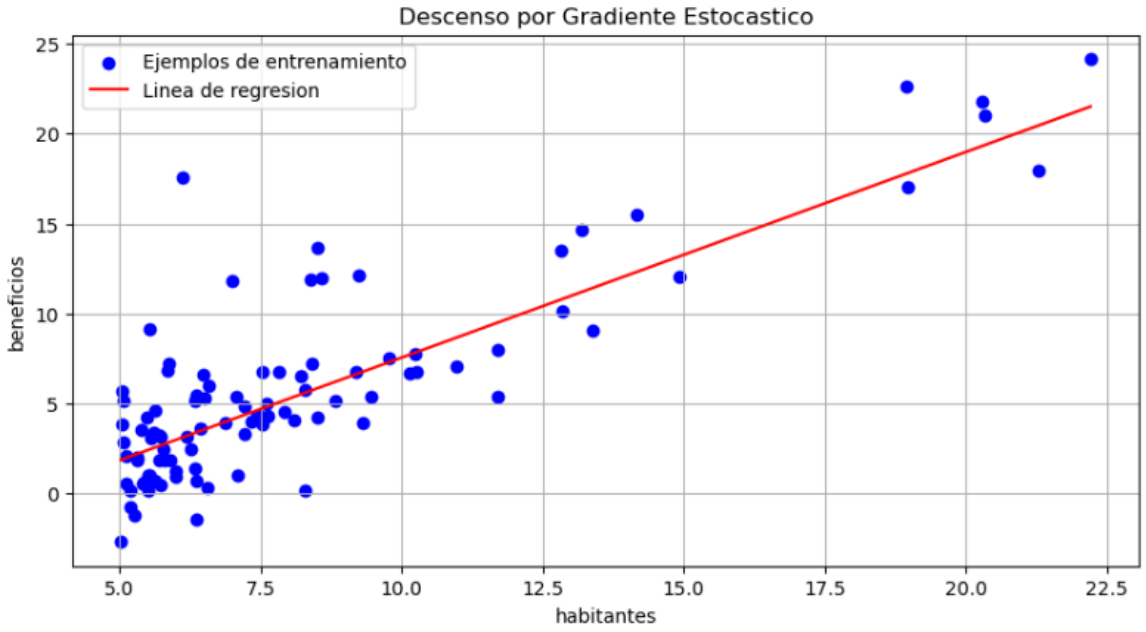
```
 resultados_tita_1.append(conj_tita_1[ind])
# obtenemos los valores de los coeficientes que minimizan el error de los 5 valores correpondientes a las distinats semillas
ind_2 = resultados_error.index(min(resultados_error))
coeficiente_0 = resultados_tita_0[ind_2]
coeficiente_1 = resultados_tita_1[ind_2]
pintar_error(conj_errores)
print(f"Coeficientes de la regresión: \nIntercepto (tita_0): {coeficiente_0:.4f}, Pendiente (tita_1): {coeficiente_1:.4f}")
plt.figure(figsize=(10, 5))
plt.scatter(x, y, color='blue', label='Ejemplos de entrenamiento')
x_linea = np.linspace(min(x), max(x), 1000)
y_linea = coeficiente_0 + coeficiente_1 * x_linea
plt.plot(x_linea, y_linea, color='red', label='Linea de regresion')

plt.title('Descenso por Gradiente')
plt.xlabel('habitantes')
plt.ylabel('beneficios')
plt.legend()
plt.grid(True)
plt.show()
```

6. Descenso por gradiente estocástico:

El concepto estocástico consiste en calcular mediante la formula del gradiente los valores de la regresión sobre un numero de ejemplos (batch) elegidos al azar no sobre todos los ejemplos, eso implica que alcanza la convergencia mucho antes que el descenso por gradiente, este algoritmo nos propone una solución, pero no la óptima dado que escoge los datos al azar.

En la siguiente grafica observamos los valores de los coeficientes con la recta de la regresión:

```
Coeficientes de la regresión:
Intercepto (tita_0): -3.8755, Pendiente (tita_1): 1.1433
```



El código desarrollado es el siguiente:

```
# Descenso por gradiente estocastico
alfa = 0.0001 # inicializamos la tasa de aprendizaje
conj=[123456789,234567890,345678900,456789000,567890000] # conjuntos que contiene las semillas que usaremos
resultados_error = []
resultados_tita_0 = []
resultados_tita_1 = []
batch = 50 # numero de valores que usamos para el calculo de la formula del gradiente estocastico
for k in conj:
 # diferentes semillas
 random.seed(k)
 # incializamos las titas aa valores aleatorios entre 0 y 1
 tita_0 = random.random()
 tita_1 = random.random()
 conj_errores = []
 conj_tita_0 = []
 conj_tita_1 = []
 error = 0
 for i in range(5000): # numero de iteraciones para el criterio de parada (hasta que converga) al ser
                       # aleatorio aumentamos el numero de iteraciones para ver si se puede tener mejor solucion
   sum_0, sum_1, error = 0, 0,0
   conj_batch=[]
   for h in range(batch): # obtener 10 datos de los ejemplos para el caluclo del gradiente estocastico
     num = random.randint(0,len(x)-1)
     conj_batch.append((x.iloc[num], y.iloc[num]))
   for j in conj_batch: # caluclar el sumatorio de gradiente estocastico del conjunto obtenido antes
     h_x = tita_0 + tita_1 * j[0]
     sum_0 += (h_x - j[1])
     sum_1 += ((h_x - j[1]) * j[0])
     error += (h_x - j[1]) ** 2
   # modificamos los coeficientes restando el valor obtenido para que decrececa las titas en sentido contrario
   tita_0 -= alfa * sum_0
   tita_1 -= alfa * sum_1
  # añadimos los datos(valores de los coeficientes y el error) a los conjuntos
   conj_errores.append(0.5 * error/batch)
    conj_tita_0.append(tita_0)
    conj_tita_1.append(tita_1)
 # calculamos el valor que minimiza el error para obtener los coeficientes que le corresponden
 mini = min(conj_errores)
 ind = conj_errores.index(mini)
 resultados_error.append(mini)
 resultados_tita_0.append(conj_tita_0[ind])
 resultados_tita_1.append(conj_tita_1[ind])
# obtenemos los valores de los coeficientes que minimizan el error de los 5 valores correpondientes a las distinats semillas
ind_2 = resultados_error.index(min(resultados_error))
coeficiente_0 = resultados_tita_0[ind_2]
coeficiente_1 = resultados_tita_1[ind_2]
```

En cada iteración del algoritmo descenso por gradiente estocástico, los coeficientes se ajustan a un subconjunto aleatorio pequeño de los datos, y esto hace que las actualizaciones sean más rápidas, pero menos precisas, eso hace que se utiliza un numero de iteraciones mayor para compensar esa falta de precisión.

7. Función de predicción:

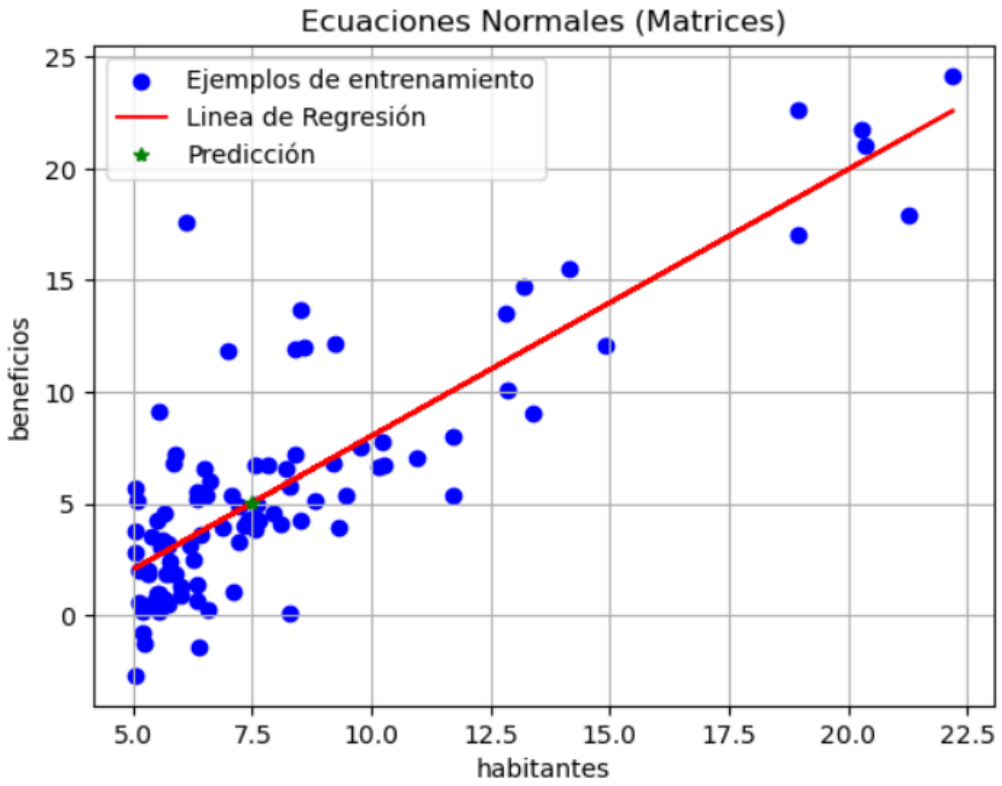
Como resultado final íbamos buscando la predicción de beneficios en función de la población, para ello hemos diseñado la siguiente función donde le pasamos los valores de la regresión obtenidas mediante alguno de los apartados anteriores y el numero de la población.

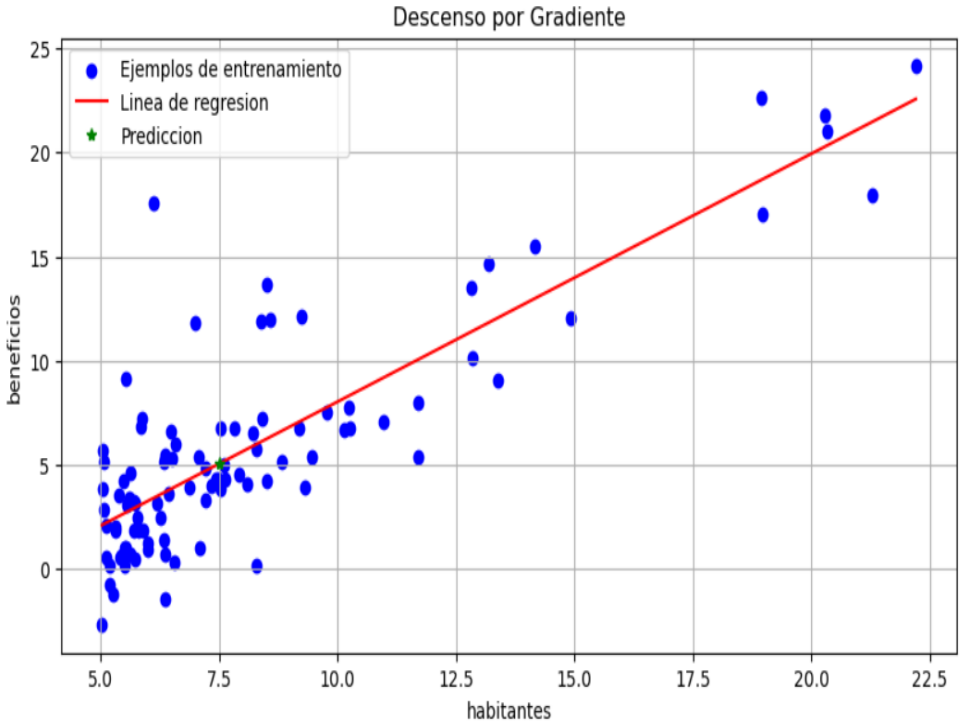
```
def calcular_Predicado(coeficiente1, coeficiente2, valor_entrada):
  return coeficiente1 + coeficiente2 * valor_entrada
```

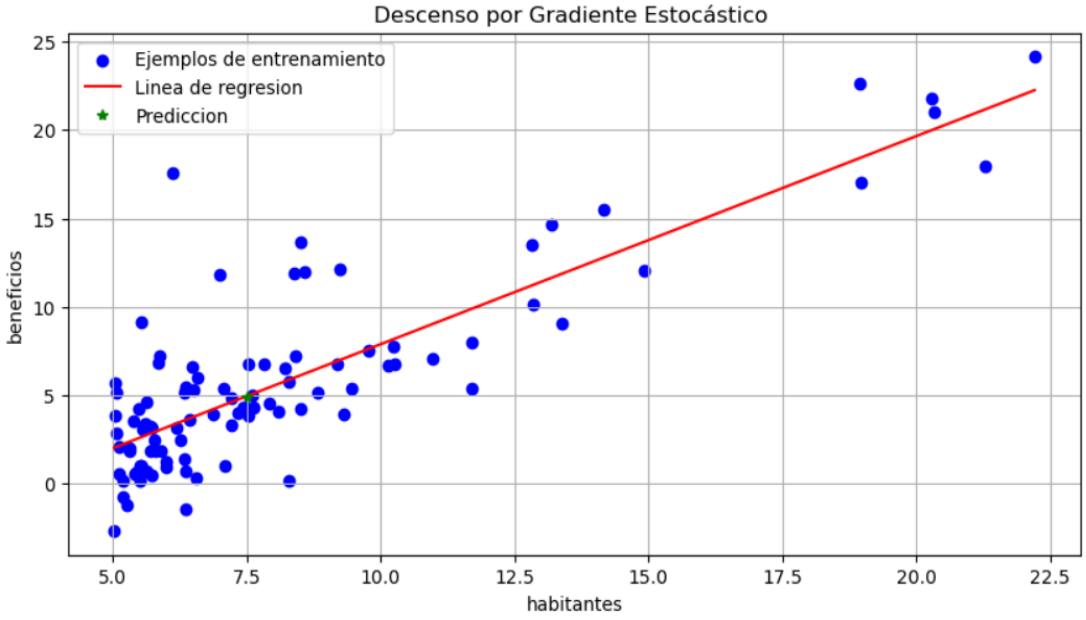
Ejemplos de la predicción con distintos valores:

```
El beneficio que corresponde a una poblacion de 9.5 personas es : 7.4380 €
El beneficio que corresponde a una poblacion de 67.8 personas es : 76.9919 €
```

8. Representación de punto con las líneas de los tres algoritmos:







Para x = 7.5, los tres algoritmos nos proporcionan un valor de predicción equivalente a 5 como se muestra en las figuras anteriores.