Tarea 1 de programación: Regresión Lineal

Modalidad de Trabajo

La tarea es de carácter individual (excepcionalmente en grupos de 2 personas, previa autorización del profesor) y no se deben compartir entre los alumnos los código, solo pueden establecer conversaciones respecto a los conceptos teóricos y prácticos asociados a la temática a ser desarrollada en la tarea.

Entrega

Jueves 29/05/2025 antes de las 15:40 hrs.

Se debe realizar un informe escrito y un programa que realice lo solicitado. El informe y el programa deben enviarse por e-mail al profesor (vladimir.riffo@uda.cl), antes de la hora acordada. No se aceptará entrega fuera de plazo. El día de la entrega se debe exponer en clase, por lo tanto, se debe traer a clase el código, así como el informe en formato electrónico para poder hacer la presentación.

La tarea deberá ser enviada en un archivo comprimido con la siguiente estructura: `Tarea1_Nombre_Apellido.rar` y al interior del archivo comprimido, debiesen estar todos los códigos implementados, señalando al código principal con el nombre de `main.py` o `main.ipynb` u otro compatible con Python.

Al usuario del código (profesor) no se le debe pedir que ingrese manualmente ningún dato de entrenamiento, solo en la prueba del modelo, se le debe solicitar el ingreso de un dato, para que el algoritmo haga la estimación. Se evaluará positivamente la usabilidad del código.

Informe Escrito:

Debe ser escrito en Times New Roman, tamaño 12, espacio simple. El informe escrito debe tener carátula (tapa), cuerpo y opcionalmente anexos que contenga el código de los programas. El informe debe contener:

0. Índice de Contenidos
1. Introducción (se muestra el contexto, necesidad, tarea, descubrimientos, conclusiones y perspectivas. Debiese además, incluir referencias bibliográficas).
2. Marco Teórico (se explica la teoría empleada, con las respectivas referencias bibliográficas)
3. Diseño (se explica qué se hizo)
4. Experimentos y resultados
5. Conclusiones
6. Bibliografía

Anexo A: Manual de uso del programa

El cuerpo del informe (puntos 1, 2, ... 6) debe ser de 4 a 6 páginas como máximo.

1. Objetivo

Implementar en Python la regresión lineal y analizar cómo funciona con los datos.

2. Regresión lineal con una variable

Antes de comenzar: debe descargar el archivo de datos `data1.txt` y estudiar su estructura.

En esta parte de la tarea, implementará la regresión lineal con una variable para predecir las ganancias de un camión de comida (food truck). Suponga que es el director ejecutivo de una franquicia de restaurantes y están considerando diferentes ciudades para abrir un nuevo establecimiento. La cadena ya tiene camiones en varias ciudades y tienes datos de ganancias y poblaciones de las ciudades.

Le gustaría utilizar estos datos para ayudarlo a seleccionar a qué ciudad expandirse a continuación.

El archivo `data1.txt` contiene el conjunto de datos para nuestro problema de regresión lineal. La primera columna es la población de una ciudad y la segunda columna es el beneficio de un camión de comida en esa ciudad. Un valor negativo de la utilidad indica una pérdida.

2.1 Trazar los datos

Antes de comenzar con cualquier tarea, a menudo es útil comprender los datos visualizándolos. Para este conjunto de datos, puede usar un diagrama de dispersión para visualizar los datos, ya que solo tiene dos propiedades para trazar (beneficio y población). (Muchos otros problemas que encontrará en la vida real son multidimensionales y no se pueden trazar en una trama bidimensional).

El resultado final debería parecerse a la Figura 1, con los mismos marcadores "×" rojos y etiquetas de eje.

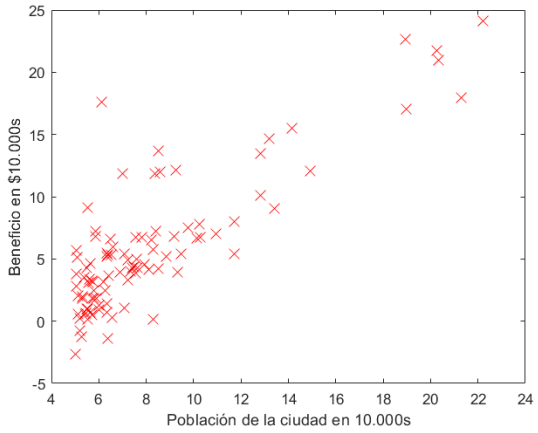


Figura 1: Diagrama de dispersión de datos de entrenamiento.

2.2 Gradiente descendente

En esta parte, ajustará los parámetros de regresión lineal θ a nuestro conjunto de datos usando el gradiente descendente.

2.2.1 Actualizar ecuaciones

El objetivo de la regresión lineal es minimizar la función de costo:

$$J(\theta_0, \theta_1) = J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

donde la hipótesis $h_\theta(x)$ viene dado por el modelo lineal:

$$h_\theta(x) = \theta^\top x = \theta_0 + \theta_1 x_1$$

Recuerde que los parámetros de su modelo son los θ_j valores. Estos son los valores que ajustará para minimizar el costo $J(\theta)$. Una forma de hacer esto es utilizar el algoritmo de gradiente descendente por lotes. En el gradiente descendente por lotes, cada iteración realiza la actualización:

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x_j^{(i)} \quad ; (\text{adaptar simultaneamente } \theta_j \text{ para } j = 0, 1)$$

Con cada paso del gradiente descendente, sus parámetros θ_j se acercarán a los valores óptimos que lograrán el menor costo $J(\theta)$.

> **Nota de implementación:** Almacenar cada ejemplo como una fila en la matriz *X* de Python. Hay que tener en cuenta el término de intersección θ_0, agregando una primera columna adicional a *X* y configurarla toda en 1s. Esto nos permite tratar θ_0 simplemente como otra "característica".

2.2.2 Implementación

Hay que inicializar los parámetros θ_0 y θ_1 iniciales a 0 y la tasa de aprendizaje `alfa` a 0,01.

2.2.3 Calcular el costo $J(\theta)$

A medida que se ejecuta el gradiente descendente para aprender a minimizar la función de costo $J(\theta)$, Es útil monitorear la convergencia calculando el costo. En esta parte, se debe implementar una función para calcular $J(\theta)$ para que pueda comprobar la convergencia de su implementación del gradiente descendente. Lo próximo que se debe implementar es una función que calcule $J(\theta)$. Mientras hace esto, recuerde que las variables *X* e *y* no son valores escalares, sino matrices cuyas filas representan los ejemplos del conjunto de entrenamiento.

En pseudocódigo:

```
función J ← computeCost(X, y, theta)
   m ← largo(y)
   J ← (1/(2*m))*suma((X*theta-y).^2)  #;.^2 significa que cada elemento es elevado a 2.
fin_función
```

Una vez que hayas completado la función, el siguiente paso es ejecutar la función `computeCost` una vez, usando los θs inicializados a ceros, y debe mostrar el costo en la pantalla. Debería esperar ver un costo de 32,07.

Ahora debe enviar "calcular el costo" para la regresión lineal con una variable.

2.2.4 Gradiente Descendente

A continuación, implementarás el gradiente descendente, siguiendo el pseudocódigo:

```
función [theta, J_historia] = gradientDescent(X, y, theta, alpha, num_iters)
m ← largo(y)
J_historia ← zeros(num_iters, 1)
for iter = 1:num_iters
    theta0 ← theta(1)-alpha*(1/m)*suma((X*theta-y).*X(:,1))   #; .* es la multiplicación "uno a uno"
    theta1 ← theta(2)-alpha*(1/m)*suma((X*theta-y).*X(:,2))
    theta ← [theta0; theta1]

   J_historia(iter) = computeCost(X, y, theta)
fin_for
fin función
```

Mientras programa, asegúrese de comprender lo que está tratando de optimizar y lo que se está actualizando. Considere probar con `num_iters = 1500`. Tenga en cuenta que el costo $J(\theta)$ está parametrizado por el vector θ, no por X e y. Es decir, minimizamos el valor de $J(\theta)$ cambiando los valores del vector θ, no cambiando X o y.

Una buena forma de verificar que el descenso del gradiente funciona correctamente es observar el valor de $J(\theta)$ y compruebe que está disminuyendo con cada paso. En cada iteración de ejecución de `computeCost` imprima el costo. Suponiendo que ha implementado el gradiente descendente y `computeCost` correctamente, su valor de $J(\theta)$ nunca debería aumentar y debería converger a un

valor estable al final del algoritmo.

Una vez que haya terminado, el algoritmo principal utilizará sus parámetros finales para trazar el ajuste lineal. El resultado debería parecerse a la Figura 2:

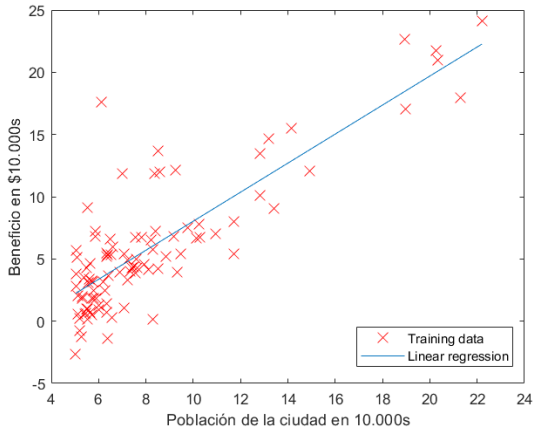


Figura 2: Datos de entrenamiento con ajuste de regresión lineal

Nota: Para cada iteración debe desplegar sobre la Figura 2 el ajuste de regresión lineal, de tal forma de visualizar que tan bien funciona el algoritmo.

Tus valores finales para θ también se utilizará para **hacer predicciones sobre beneficios en áreas de 35.000 y 70.000 habitantes**.

2.3 Visualización $J(\theta)$

En esta parte de la ejecución, podrá probar diferentes tasas de aprendizaje para el conjunto de datos y encontrar una tasa de aprendizaje que converja rápidamente. Puede cambiar la tasa de aprendizaje modificando la parte del código que establece la tasa de aprendizaje. La siguiente fase en el algoritmo principal llamará a la función `gradientDescent` y ejecutara el gradiente descendente durante aproximadamente 50 iteraciones a la velocidad de aprendizaje elegida. La función también debería devolver el historial de $J(\theta)$ valores en un vector `J`. Después de la última iteración, debes trazar los `J` valores en función del número de iteraciones.

Si eligiste una tasa de aprendizaje dentro de un buen rango, su diagrama se verá similar a la Figura 3. Si su gráfico se ve muy diferente, especialmente si su valor de $J(\theta)$ aumenta o incluso explota, ajuste su tasa de aprendizaje y vuelva a intentarlo. Recomiendo probar valores de la tasa de aprendizaje α en una escala logarítmica, en pasos multiplicativos de aproximadamente 3 veces el valor anterior (es decir, 0.3, 0.1, 0.03, 0.01 y así sucesivamente). También es posible que desee ajustar el número de iteraciones que está ejecutando si eso le ayudará a ver la tendencia general en la curva.

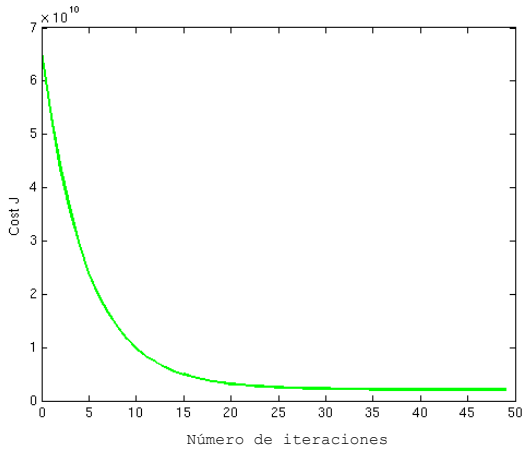
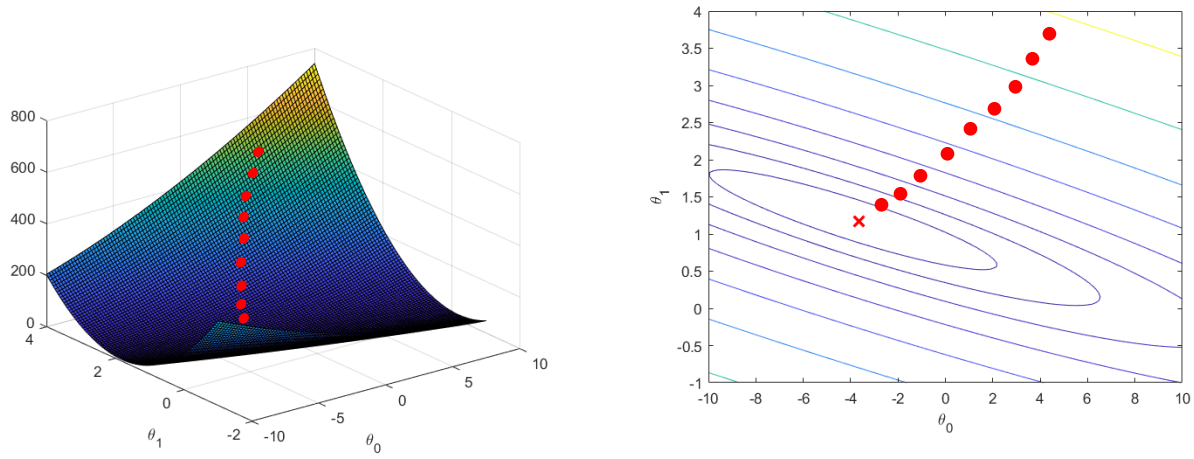


Figura 3: Convergencia del gradiente descendente con una tasa de aprendizaje adecuada.

Para comprender mejor la función de costos $J(\theta)$, ahora debe graficar el costo sobre una cuadrícula bidimensional (2D) de θ_0 y θ_1 valores y cómo en cada iteración se aproximan los valores de θ_0 y θ_1 al optimo, de tal forma que se aprecie el trabajo del gradiente descendente. Lo mismo puede ser visualizado en un gráfico 3D.



(a) (b)

Figura 4: Gráfico mostrando el mínimo y evolución del gradiente descendente. (a) de Superficie, y (b) de Contorno.

Nota: Ver el siguiente link, para ver como graficar la función de costo en 3D.

https://datascience.stackexchange.com/questions/28411/how-to-plot-cost-versus-number-of-iterations-in-scikit-learn

2.4 Ecuación Normal: $\theta = (X^\intercal X)^{-1} X^\intercal y$

Tendrá una puntuación adicional el alumno que implemente la ecuación normal y obtenga los θ_s