Practica de Regresión Lineal

El objetivo de esta práctica es que los alumnos se familiaricen con el entorno de desarrollo MATLAB, experimenten con un algoritmo de aprendizaje supervisado (regresión lineal) y comprendan el método de descenso del gradiente. Los alumnos contarán con dos scripts de ayuda (`ex1.m` y `ex1_multi.m`) que irán ejecutando, por partes, en cada apartado de la práctica, y les ayudará a comprobar si su código proporciona resultados correctos.

1. Regresión Lineal con una variable

Disponemos de un conjunto de datos que contiene, para una cierta cadena de tiendas, el beneficio de cada tienda asociado al tamaño de la ciudad donde se encuentra. La regresión lineal permitirá predecir, en función del tamaño de la ciudad, el beneficio que podemos obtener si decidimos abrir una nueva tienda en cualquier ciudad.

1.1. Representación de los datos

Puedes comprobar que `ex1.m` carga los datos de un fichero en las variables X e y:

```
data = load('ex1data1.txt');
X = data(:, 1); y = data(:, 2);
m = length(y);
```

A continuación el script hace una llamada a `plotData.m`. Debes completar `plotData.m` para que genere la gráfica. Modifica el fichero introduciendo el siguiente código:

```
plot(x, y, 'rx', 'MarkerSize', 10);
ylabel('Beneficio en 10,000s');
xlabel('Población de la ciudad en 10,000s');
```

1.2. Descenso del gradiente

En este apartado vamos a ajustar los parámetros (θ) de regresión lineal a los datos proporcionados, mediante el algoritmo de descenso del gradiente.

1.2.1. Ecuaciones

El objetivo de la regresión lineal es minimizar la función de coste

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

donde la hipótesis viene dada por el modelo lineal

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

Los parámetros de nuestro modelo son θ_0 y θ_1 y hay que ajustarlos hasta minimizar el valor de $J(\theta)$. Esto lo haremos mediante el algoritmo de descenso del gradiente. Este algoritmo realiza las siguientes operaciones en cada iteración:

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

para cada j (en este caso j = 0 y j = 1). En cada iteración del algoritmo de descenso del gradiente, los valores de θ_0 y θ_1 se acercan más al valor que minimiza la función $J(\theta)$.
En `ex1.m` se preparan los datos para regresión lineal: se añade una dimensión más a los datos para acomodar el término θ_0 y se inicializan los parámetros a 0 y la tasa de aprendizaje a 0.01.

```
X = [ones(m, 1), data(:,1)];
theta = zeros(2, 1);
iterations = 1500;
alpha = 0.01;
```

1.2.2. Cálculo del coste $J(\theta)$

Al realizar el descenso del gradiente conviene monitorizar la convergencia de la función de coste. **Debes implementar la función $J(\theta)$**. Para ello debes introducir el código necesario en el fichero `computeCost.m`, que es la función que devuelve el valor de $J(\theta)$. Recuerda que las Variables X e y no son escalares sino matrices cuyas filas representan los ejemplos de los datos de entrenamiento.

Una vez que hayas completado la función, el siguiente paso en `ex1.m` realizará una llamada a `computeCost` una sola vez, empleando θ inicializado con ceros, y verás el coste en la pantalla.

Debes obtener un coste *aproximadamente* igual a 32.07.

1.2.3. Descenso del gradiente

Ahora implementarás descenso del gradiente en el fichero `gradientDescent.m`. La estructura del bucle ya está escrita y **sólo debes añadir las actualizaciones a dentro de cada iteración**. Para verificar que el descenso del gradiente está funcionando correctamente el código de `gradientDescent.m` llama a `computeCost` en cada iteración y muestra el coste $J(\theta)$ por pantalla. Si la implementación es correcta el valor de $J(\theta)$ nunca debe aumentar, y debe converger a un valor estable al finalizar el algoritmo.

Cuando hayas acabado de programar, el siguiente paso de `ex1.m` hace uso de los parámetros calculados con tu código para imprimir la función lineal. Tus valores finales de θ también se emplearán para predecir los beneficios en poblaciones de 35000 y 70000 personas, mediante las siguientes líneas de `ex1.m`.

```
predict1 = [1, 3.5] * theta;
predict2 = [1, 7] * theta;
```

1.3. Visualizando $J(\theta)$

Para comprender mejor la función de coste $J(\theta)$, vamos a representar el coste en un plano bidimiensional de valores θ_0 y θ_1. En este apartado no es necesario escribir ningún código, pero debes entender el código que se proporciona.
En el siguiente paso de `ex1.m`, hay un código preparado para calcular $J(\theta)$ en un conjunto de valores θ_0 y θ_1 usando la función `computeCost`.

```
J_vals = zeros(length(theta0_vals), length(theta1_vals));
for i = 1:length(theta0_vals)
        for j = 1:length(theta1_vals)
                t = [theta0_vals(i); theta1_vals(j)];
                J_vals(i,j) = computeCost(x, y, t);
        end
end
```

Tras la ejecución de estas líneas, disponemos de una matriz de valores de $J(\theta)$. El script `ex1.m` usa estos valores para generar una superficie y un mapa de niveles de usando los comandos `surf` y `contour`. El objetivo de estos gráficos es mostrar como $J(\theta)$ varía con los cambios de θ_0 y θ_1. La función de coste $J(\theta)$ es totalmente convexa.

2. Regresión Lineal con múltiples variables

En este apartado aplicaremos regresión lineal con múltiples variables para predecir el precio de viviendas unifamiliares. Para ello aplicaremos la regresión sobre un conjunto de datos sobre viviendas que se han vendido recientemente. El fichero `ex1data2.txt` contiene un conjunto de datos organizados en 3 columnas: la primera contiene el tamaño de la vivienda, la segunda el número de dormitorios y la tercera columna el precio de la casa. Para este apartado emplearemos el script `ex1_multi.m`.

2.1. Normalización de variables

El script `ex1_multi.m` carga los datos y muestra algunos de ellos. Vemos que los tamaños de parcela son más de 100 veces mayores que el número de dormitorios. Cuando las variables difieren en varios ordenes de magnitud es conveniente escalarlas para que el descenso del gradiente converja más rápidamente.

En este apartado **debes completar el código de** `featureNormalize.m` **para que haga lo siguiente**:

- Restar a cada variable el valor medio de su columna
- Tras la resta, escalar (dividir) los valores de cada variable por su respectiva "desviación estándar".

La desviación estándar es una forma de medir cuánta variación hay en el rango de valores de una variable (la mayor parte de los puntos estarán dentro del rango de ±2 de desviación estándar). Puedes emplear la función `std` de MATLAB para calcular la desviación estándar. Por ejemplo, en `featureNormalize.m` la cantidad `X(:,1)` contiene todos los valores de tamaños de vivienda de los datos de entrenamiento, así que `std(X(:,1))` calculará la desviación estándar de los tamaños de viviendas. El código debe funcionar para cualquier tamaño del conjunto de datos (cualquier número de variables y ejemplos). Recuerda que cada columna de `X` corresponde a una variable.

2.2. Descenso del gradiente

En este apartado **debes completar el código de** `computeCostMulti.m` y `gradientDescentMulti.m` **para que implementen la función de coste y el descenso de gradiente para regresión lineal con múltiples variables.** Si tu código de la parte anterior (una variable) ya esta preparado para múltiples variables, puedes usarlo aquí

también. Asegúrate de que tu código soporta cualquier número de variables y está bien vectorizado. Puedes usar `size(X, 2)` para determinar cuántas variables están presentes en el conjunto de datos. También te puede ayudar ver como se puede escribir la función de coste de forma vectorizada:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

donde

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

2.2.1 Seleccionando las tasas de aprendizaje

En este último apartado probaremos distintas tasas de aprendizaje para los datos y buscaremos una tasa de aprendizaje que proporcione una rápida convergencia. Puedes cambiar la tasa de aprendizaje modificando `exl_multi.m` y cambiando la parte de código donde se especifica `alpha`. El siguiente paso de `exl_multi.m` llamará a la función `gradientDescent.m` y ejecutará 50 iteraciones de descenso de gradiente para la tasa seleccionada. La función devuelve la historia de valores de $J(\theta)$ en el vector `J`. Tras la última iteración, el script `exl_multi.m` imprime los valores de `J` frente al número de iteraciones.

Es recomendable escoger valores de la tasa de aprendizaje en escala logarítmica, en pasos multiplicativos de aproximadamente 3 veces el valor anterior (es decir: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3). También puedes cambiar el número de iteraciones si lo necesitas para ver la tendencia general de la curva `J`). Recuerda que si la tasa de aprendizaje es muy grande, $J(\theta)$ puede divergir y contener valores demasiado grandes para representación numérica en MATLAB (NaN puede significar tanto ∞ como −∞).

Para comparar distintas tasas de aprendizaje es útil representar `J` para distintas tasas en la misma figura. Puedes hacer esto con el comando `hold on`. Por ejemplo, si tienes 3 valores diferentes de `alpha` (aunque deberías probar con alguno más), y has guardado los costes en `J1` `J2` y `J3`, puede usar los siguientes comandos para representarlos en la misma figura, con distintos colores:

```
plot(1:50, J1(1:50), 'b');
hold on;
plot(1:50, J2(1:50), 'r');
plot(1:50, J3(1:50), 'k');
```

Con la mejor tasa de aprendizaje que hayas encontrado, emplea el script `exl_multi.m` para ejecutar el descenso de gradiente hasta que converja a un valor final de θ. A continuación emplea ese valor de θ para predecir el precio de una casa de 120 metros y 3 habitaciones. No te olvides de normalizar estos valores antes de realizar la predicción.