Aprendizaje automático y minería de datos

Práctica 1 – Regresión lineal

Ramón Arjona Quiñones
Celia Castaños Bornaechea

Objetivo

Implementar un algoritmo que predice los valores de salida utilizando unos casos de entrenamiento utilizando regresión lineal con una o más variables de entrada.

Método de carga

Carga el fichero especificado y devuelve los datos en formato array de numpy. Supondremos que dichos datos serán tipo *float*

```
def carga_csv(file_name):
    """carga el fichero csv especificado y lo devuelve en un array de num
py
    """
    valores = read_csv(file_name, header=None).values

    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

Función hipótesis

Calcula y devuelve el valor para la *Y* a partir de unos *X* (casos de entrenamiento) y *Theta* pasados como parámetros. Esto se puede llevar a cabo mediante dos fórmulas.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$h_\theta(x) = \theta^T x$$

Para llevarlo a cabo de la segunda forma debemos añadir una columna de unos al principio de la matriz *X*, esto significa que se le añade un 1 a cada ejemplo de entrenamiento.

Al ser una hipótesis se estima que la salida es la correcta pero no se tiene completa seguridad.

Nosotros hemos utilizado la segunda fórmula para llevar a cabo la práctica.

```
def h(x, theta):
    return np.dot(x, theta[np.newaxis].T)
```

Función coste

Calcula cómo de lejos está la hipótesis calculada de los ejemplos de entrenamiento. Esto nos indica cómo de buena es la función hipótesis

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

Siendo, en el código, *sum3* la variable que va guardando el valor del sumatorio en cada vuelta del bucle. La variable *m* es la cantidad de casos de entrenamiento dados y, por lo tanto, el número de iteraciones del sumatorio.

```
def coste(X, Y, Theta):
    #Calculamos la función de coste
    m = X.shape[0]
    sum3 = 0
    for k in range(0, m):
        sum3 += (h(X[k], Theta) - Y[k])**2
    return 1 / (2*m) * sum3
```

Descenso de gradiente

Es un algoritmo de optimización para encontrar los parámetros de nuestro modelo que mejor definan el conjunto de entrenamiento. Nos vamos acercando iterativamente a un valor que minimiza la función de coste.

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

Siendo *alpha* un número dado por parámetros para buscar el punto mínimo. Debe tener un valor muy pequeños puesto que si es muy grande existe la posibilidad de pasarse el mínimo.

En este método tiene lugar la adición de la columna de unos a la matriz *X* y la inicialización del vector *theta* a cero. *Theta* se va actualizando para obtener unos valores aproximados que minimicen la función de coste.

```
def descenso_gradiente(X, Y, alpha, num_iter):

    m = X.shape[0]
    n = X.shape[1]
    unos = np.ones((m, 1))
    X = np.hstack((unos, X))
    #2.Calculamos la recta
    theta = np.zeros(n + 1, dtype=float)
    #Bucle
    htheta = np.zeros((m, 1))

    costes = np.ones(num_iter)
    #Cada iteración del proceso
    for i in range (num_iter):
        #Calcular el valor de h(theta) con la theta de la iteración anter
ior
        htheta = h(X, theta)

        #Calculamos el valor de las tetas
        for k in range(0, n + 1):
            sum1 = 0
            for l in range(0, m):
                sum1 += (htheta[l] - Y[l]) * X[l, k]
            theta[k] = theta[k] - alpha * (1 / m) * sum1

        #Calculamos la función de coste
```

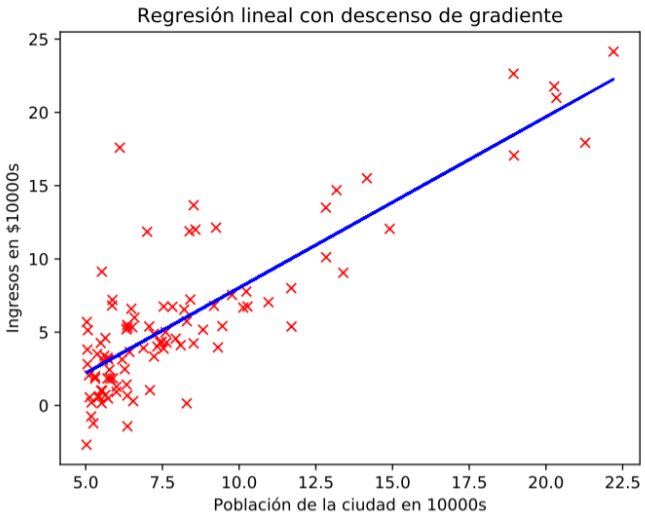
```
        H = np.dot(X, theta)
        Aux = (H-Y)**2
        costes[i] = Aux.sum() / (2*m)

    return [theta, costes]
```

Respresentación de la recta

Representa el coste, aplicando el método de descenso de gradiente, como una recta. Siendo los ejes de la gráfica la entrada (*X*) y la salida (*Y*).

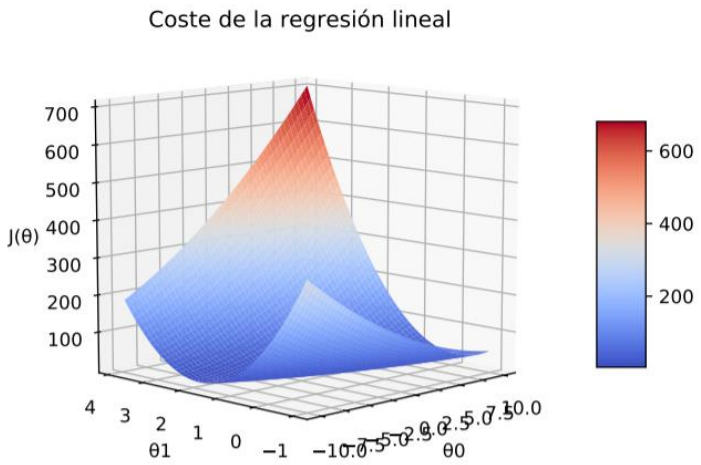
Para el ejemplo de la imagen se han llevado a cabo 1500 iteraciones y se le ha dado a *alpha* un valor de 0,01.




```
## Dibuja la gráfica con la recta de regresión
def drawLinearRegression(X, Y, f, Xlabel, Ylabel):
    plt.figure()
    plt.plot(X, Y, 'x',color="red")
    plt.plot(X, f, color="blue")
    plt.title("Regresión lineal con descenso de gradiente")
    plt.xlabel(Xlabel)
    plt.ylabel(Ylabel)
```

Representación de la superficie de coste

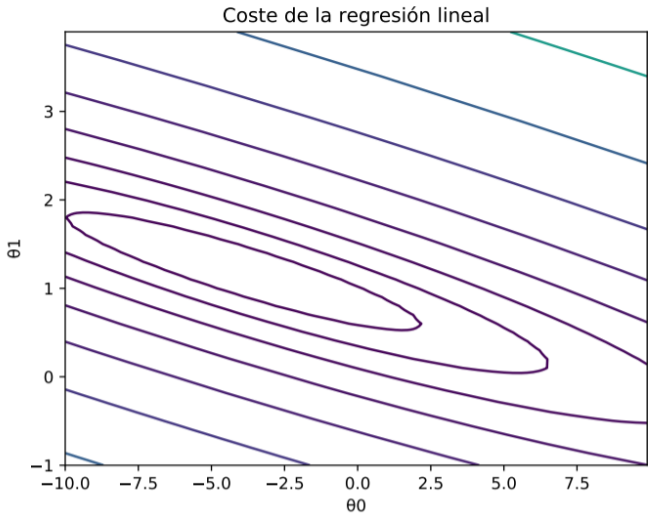
Es otra forma de representar la función de coste. Donde los ejes *X* y *Z* corresponden a los valores del vector *theta* y el eje *Y* al coste.




```
## Dibuja la superficie de coste
def drawCostSurface(Theta0, Theta1, Coste):
    #Dibujamos la gráfica de coste con las thetas
    fig = plt.figure()
    ax = fig.gca(projection = '3d')
    #Plot de surface
    surf = ax.plot_surface(Theta0, Theta1, Coste, cmap = cm.coolwarm, lin
ewidth = 0, antialiased = False)
    ax = fig.gca(projection = '3d')
    ax.set_xlabel("θ0")
    ax.set_ylabel("θ1")
    ax.set_zlabel("J(θ)")
    plt.title("Coste de la regresión lineal")
    fig.colorbar(surf, shrink = 0.5, aspect = 5)
    ax.view_init(elev=10, azim=-135) #Para rotar la figura en el eje Z
```

Representación del contorno del coste

Otra forma de visualizar el coste en una gráfica. En este caso se utiliza una escala logarítmica para el eje *Z*, donde se representa el coste.

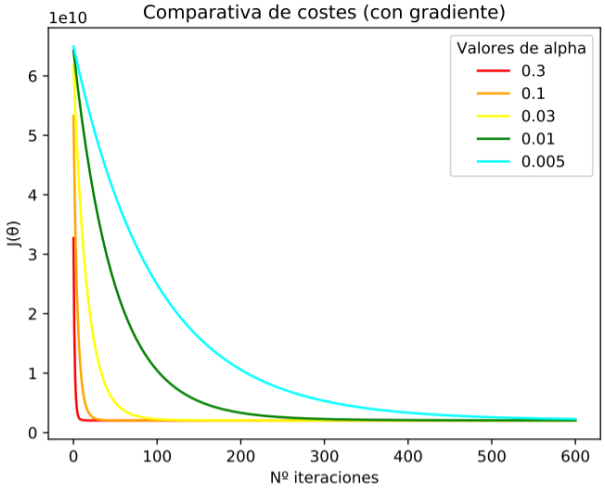



```
## Dibuja el contorno del coste
def drawContour(Theta0, Theta1, Coste):
    plt.figure()
    plt.xlabel("θ0")
    plt.ylabel("θ1")
    plt.title("Coste de la regresión lineal")
    (Theta0, Theta1, Coste, np.logspace(-2, 3, 20))
```

Representación relación *alpha* y coste

Dibuja en la gráfica cómo va disminuyendo el coste respecto al número de iteraciones.

En la imagen de ejemplo se comparan el resultado del coste respecto a los distintos valores de *Alpha.*




```
## Dibuja la gráfica de relación alpha/coste
def drawLearningRate(alpha, costes, kolor):
    size = costes.shape[0]
    X = np.linspace(0, size, size)
```

```
    plt.plot(X, costes, color=kolor, label=alpha)

## Dibuja varias learningRate
def drawAlphasCost(alphas, X, Y, num_iter, colors):
    plt.figure()
    for i in range (len(alphas)):
        thetas, costes = descenso_gradiente(X, Y, alphas[i], num_iter)
        drawLearningRate(alphas[i], costes, colors[i])
    plt.title("Comparativa de costes (con gradiente)")
    plt.xlabel("Nº iteraciones")
    plt.ylabel("J(θ)")
    plt.legend(loc='upper right', title="Valores de alpha")
```

Normalización de *X*

Se normalizan los valores de la matriz de los casos de entrenamiento, para así acelerar la convergencia en el descenso de gradiente. También se calculan *mu* (media de cada atributo) y *sigma* (desviación estándar de cada atributo).

```
## Normaliza los valores de la X (si recibe mu y sigma, lo normaliza
respecto a esos)
def normaliza(X, mu=None, sigma=None):
    # Nos pasan mu y sigma existentes
    if(mu is None or sigma is None):
        mu = np.mean(X, axis=0)
        sigma = np.std(X, axis=0)

    # Calculamos el valor normalizado
    X = (X - mu) / sigma
    return X, mu, sigma
```

Ecuación normal

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

La ecuación normal obtiene el valor óptimo de *theta* en tan solo un paso.

```
## Calcula el vector de pesos y coste mediante la ecuación normal
def normal(X, Y):

    m = X.shape[0]
    n = X.shape[1]
    unos = np.ones((m, 1))
    X = np.hstack((unos, X))

    #2.Calculamos la recta
    theta = np.zeros(n + 1, dtype=float)

    #Calculamos theta
```

```
    aux = np.linalg.pinv(np.matmul(X.T, X))
    theta = np.matmul(np.matmul(aux, X.T), Y)

    #Calculamos la función de coste
    H = np.dot(X, theta)
    Aux = (H-Y)**2
    coste = Aux.sum() / (2*m)

    return [theta, coste]
```

Ejercicio 1

Partiendo de los datos del fichero *ex1data1.csv* se aplica el método de regresión lineal sobre ellos.

Siendo la primera columna la población de una ciudad y la segunda los beneficios de una compañía que distribuye comida en dicha ciudad.

Tan solo habrá una variable de entrada y una de salida

```
## Regresión lineal con una variable mediante descenso de gradiente
def Ejercicio1(alpha, num_iter):
    #1.Importamos los datos a una matriz y rellenamos con 1's la primera columna para el p.escalar
    valores = carga_csv("ex1data1.csv")
    X = valores[:, :-1]
    Y = valores[:, -1]

    m = X.shape[0]
    unos = np.ones((m, 1))

    # Hacemos el descenso de gradiente
    theta, costes = descenso_gradiente(X, Y, alpha, num_iter)
    f = h(np.hstack((unos, X)), theta)

    ## Gráficas:
    # 1. Gráfica de la regresión lineal
    drawLinearRegression(X, Y, f, "Población de la ciudad en 10000s", "Ingresos en $10000s")
    plt.savefig('linearRegression.pdf')
    plt.close()

    # 2. Gráfica para observar la relación entre el vector de pesos y el coste de la regresión
    Theta0, Theta1, coste = make_data([-10, 10], [-1, 4], X, Y)
```

```
    drawCostSurface(Theta0, Theta1, coste)
    plt.savefig('costSurface.pdf')
    plt.close()

    # 3. Contorno de la gráfica anterior
    drawContour(Theta0, Theta1, coste)
    plt.savefig('costContour.pdf')
    plt.close()
```

Ejercicio 2

Aplicar el método de regresión lineal a los datos del archivo ex1data2.csv.

Los datos corresponden a los pies cuadrados, número de habitaciones y precio de casas vendidas en Portland, Oregón.

Por lo tanto serán dos variables de entrada y una de salida.

```
# Regresión lineal multivariable mediante normalización
def Ejercicio2(alpha, num_iter):
    ## 1.Importamos los datos a una matriz y los separamos en X e Y
    valores = carga_csv("ex1data2.csv")
    X = valores[:, :-1]
    Y = valores[:, -1]
    m = X.shape[0]
    unos = np.ones((1,))

    # Valores para los alphas y los colores que usaremos
    alphas = [0.3, 0.1, 0.03, 0.01, 0.005]
    colors = ["red", "orange", "yellow", "green", "cyan", "blue"]

    # Normalizamos los elementos de la muestra
    Xnorm, muX, sigmaX = normaliza(X)

    # Calculamos theta y el coste con ambas formas
    theta, coste = normal(X, Y) #ec.normal
    thetas, costes = descenso_gradiente(Xnorm, Y, alpha, num_iter) #gradi
ente

    # Dibujamos la comparativa de costes con los distintos valores de alp
ha
    drawAlphasCost(alphas, Xnorm, Y, num_iter, colors)
    plt.savefig("gradientCost_" + str(num_iter) + "iter.pdf")
    plt.close()
```

```
    ### 2. Ejemplo para comparar el modelo del gradiente con el normaliza
do
    ejemplo = np.array([1650, 3])
    ejemploNorm, mu, sigma = normaliza(ejemplo, muX, sigmaX)

    # Hacemos una comparativa de la forma normalizada vs el descenso de g
radiente
    prediccionNormal = h(np.hstack((unos, ejemplo)), theta)[0]
    prediccionGradiente = h(np.hstack((unos, ejemploNorm)), thetas)[0]
    dif = round((1 - abs(prediccionNormal / prediccionGradiente)) * 100,
4) #Porcentaje de diferencia

    # Dibujamos la comparativa con el ejemplo
    ejemplo = ejemplo[0] / ejemplo[1] #Para unificar los 2 atributos y po
der pintarlo en una gráfica normal
    plt.figure()
    plt.plot(ejemplo, round(prediccionGradiente / 100000, 3), 'x',color="
green", label="Gradiente")
    plt.plot(ejemplo, round(prediccionNormal / 100000, 3), 'x',color="red
", label="Normal")
    plt.legend(loc="upper right", title="Predicciones")
    plt.title("Comparativa de la predicción")
    plt.xlabel("Tamaño medio de habitación (pies cuadrados)")
    plt.ylabel("Coste en 100000$")
    plt.title(str(dif) + "% de diferencia")

    plt.savefig('normalizeExample.pdf')
    plt.close()
```