Práctica 5: Regresión lineal regularizada: sesgo y varianza

Material proporcionado:

Fichero	Explicación
`ex5data1.mat`	Conjunto de datos.

1. Regresión lineal regularizada

El objetivo de esta práctica es comprobar los efectos del sesgo (*bias*) y la varianza (*variance*). Aplicaremos regresión lineal regularizada para aprender una hipótesis sesgada, que no es capaz de clasificar correctamente a los ejemplos de entrenamiento, y a continuación usaremos de nuevo la regresión lineal para sobre-ajustar los datos de entrenamiento a un polinomio de grado superior.

Para empezar, has de cargar el fichero de datos `ex5data1.mat` que contiene datos históricos sobre el agua que ha derramado una presa en base a los cambios en el nivel del agua. Una vez cargado, con la función `scipy.io.loadmat` puedes comprobar que el diccionario resultante de la carga incluye las claves: `X` e `y`, con los datos de entrenamiento; `Xval` e `yval` con los ejemplos de validación; y `Xtest` e `ytest` con los de prueba.

A continuación, has de implementar (mejor si es forma vectorizada), una función que devuelva el coste y el gradiente de la regresión lineal regularizada a partir de los valores de entrada `X`, salida `y`, el vector de parámetros θ y el valor de λ, y recuerda que el coste viene dado por la expresión:

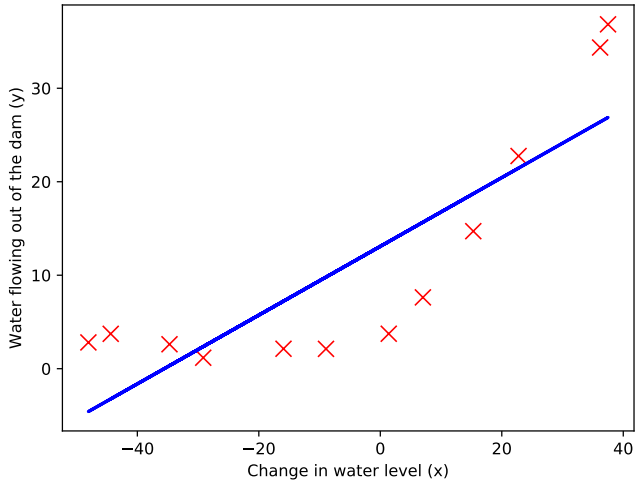
$$J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right) + \frac{\lambda}{2m}\left(\sum_{j=1}^{n}\theta_j^2\right)$$

y el gradiente por:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_0} &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} && \text{para } j = 0 \\ \frac{\partial J(\theta)}{\partial \theta_j} &= \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j && \text{para } j \geq 1 \end{aligned}$$

donde en el término de regularización, tanto del coste como del gradiente, no se incluye θ_0, la primera componente del vector θ. Para un valor de $\lambda = 1$ y $\theta = [1; 1]$ deberías obtener aproximadamente un coste de 303,993 y un gradiente de $[-15{,}303; 598{,}250]$.

A continuación, utiliza la función `scipy.optimize.minimize` para encontrar el valor de θ que minimiza el error sobre los ejemplos de entrenamiento. En el entrenamiento, fija el valor de $\lambda = 0$ pues con tan pocos parámetros no tiene mucho sentido añadir el término de regularización. El resultado, ha de ser un vector θ que defina una recta ajustada a los datos de `X` e `y` similar a esta:



2. Curvas de aprendizaje

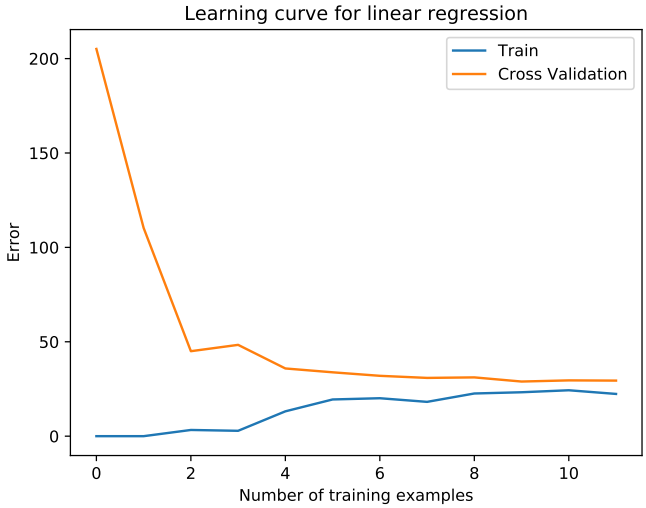
El problema del resultado obtenido en el apartado anterior es que la representación elegida para las hipótesis, una recta, es demasiado simple para ajustarse a los datos de entrenamiento y por ello predice valores sesgados a la recta. Aunque en este caso resulta muy sencillo de observar en la gráfica, cuando los ejemplos vienen descritos por más atributos no es posible tener una representación gráfica tan evidente. En ese caso se utilizan las curvas de aprendizaje para identificar situaciones de sesgo (sub-ajuste) y varianza (sobre-ajuste).

Para generar las curvas de aprendizaje, has de repetir el entrenamiento por regresión lineal del apartado anterior, utilizando diferentes subconjuntos de los datos de entrenamiento. Primero, 1, después 2, y así sucesivamente hasta llegar a m (en python, puedes generar el subconjunto de tamaño i como `X[0: i]`, `y[0:i]`.

Una vez realizado el entrenamiento por regresión lineal para ajustarse al subconjunto de entrenamiento `X[0: i]`, `y[0:i]`, has de evaluar el error del resultado aplicado sobre ese mismo subconjunto, así como el error al clasificar a todos los ejemplos del conjunto de validación (`Xval` e `yval`). Recuerda que el error de una hipótesis h_θ sobre un conjunto de ejemplos viene dado por la expresión:

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

De esta forma, obtendrás m valores para el error sobre los subconjuntos de los ejemplos de entrenamiento y el error sobre los ejemplos de validación que deberían dar lugar a una representación gráfica similar a esta:



El hecho de que en esta curva el error al aumentar el número de ejemplos de entrenamiento se aproxime en los conjuntos de entrenamiento y validación indica que el aprendizaje está sesgado y es necesario utilizar una hipótesis más expresiva que sea capaz de ajustarse mejor a los ejemplos de entrenamiento.

3. Regresión polinomial

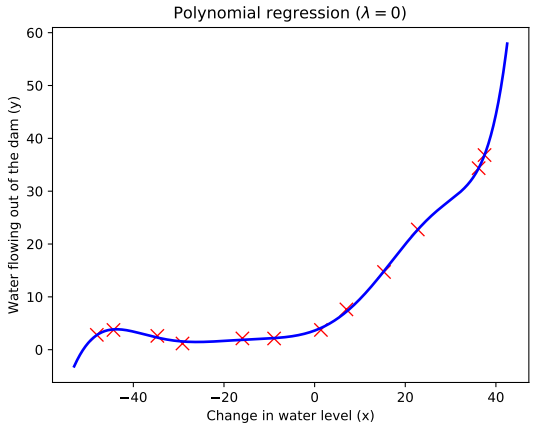
Para conseguir un mayor ajuste a los datos de entrenamiento, usaremos como hipótesis un polinomio de la variable de entrada x que representa el nivel de agua en la presa:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 * (\text{nivelAgua}) + \theta_2 * (\text{nivelAgua})^2 + \ldots + \theta_p * (\text{nivelAgua})^p \\ &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p \end{aligned}$$

Para ello, en primer lugar, has de implementar una función que sea capaz de generar los nuevos datos de entrenamiento a partir de los datos originales `X`. Esta función recibirá una matriz `X` de dimensión $m \times 1$ y un número `p`, y devolverá otra matriz de dimensión $m \times p$ que en la primera columna contenga los valores de `X`, en la segunda el resultado de calcular `X.^2`, en la tercera `X.^3`, y así sucesivamente.

El problema de generar nuevos atributos de esta forma es que se introducen grandes diferencias de rango (por ejemplo, si $x = 40$ entonces $x_8 = 40^8 = 6{,}5 \times 10^{12}$), por lo que es necesario normalizar los atributos antes de realizar el aprendizaje de θ (la normalización de los atributos es un paso previo aconsejable en la mayoría de los casos). Para ello, debes implementar una función que recibe una matriz de dimensión $m \times p$ y devuelve: otra matriz de la misma dimensión normalizada columna por columna a media 0 y desviación estándar 1 (es decir, interpretando que cada columna de la matriz de entrada representa los valores de un mismo atributo); un vector de dimensión $1 \times p$ con la media de cada columna (`numpy.mean`); y otro vector también de dimensión $1 \times p$ con la desviación estándar (`numpy.std`) de cada columna de la matriz de entrada.

Genera de esta forma los nuevos datos de entrada para aprender un polinomio de grado $p = 8$ (a los que has de añadir una columna de 1's) y vuelve a aplicar el método de regresión lineal para obtener el vector θ que minimiza el error, para un valor de $\lambda = 0$. El valor obtenido para θ debería generar una curva similar a esta:



donde se ha utilizado el resultado del entrenamiento para predecir los valores de y para un conjunto de valores x entre el mínino y el máximo de los valores de entrenamiento dados por X, a intervalos de 0,05. Para hacer predicciones sobre nuevos valores de x es necesario aplicarles la misma transformación que a los de entrenamiento, generando las potencias desde 1 hasta p y normalizándolas luego usando las medias y desviaciones estándar calculadas para los ejemplos de entrenamiento X.

A continuación, debes generar las curvas de aprendizaje para la hipótesis polinomial, utilizando la misma técnica del apartado anterior que consiste en aplicar la regresión a subconjuntos cada vez mayores de los datos de entrenamiento y evaluar el error de la hipótesis aprendida sobre ese subconjunto y un conjunto independiente de ejemplos de validación. De nuevo, no olvides que para calcular el error sobre los ejemplos de validación `Xval` debes aplicarles la misma transformación que a los de entrenamiento. El resultado debería ser similar a este (puede depender del número de iteraciones del método de minimización[1]):

donde se muestra que la hipótesis está sobre-ajustada a los ejemplos de entrenamiento sobre los que presenta un error de 0. A continuación, puedes probar el efecto que tiene el término de regularización repitiendo el proceso para $\lambda = 1$ y $\lambda = 100$.

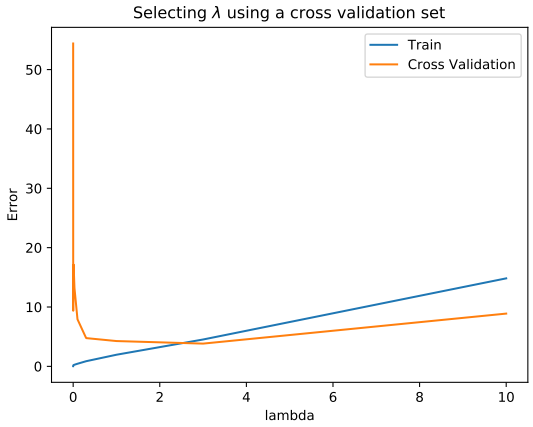
[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

4. Selección del parámetro λ

Como debes haber comprobado, el parámetro λ del término de regularización permite controlar el grado de ajuste a los ejemplos de entrenamiento. Pero, ¿cuál es el mejor valor posible para λ?

La técnica que se suele utilizar para elegir el valor de λ es evaluar la hipótesis generada sobre los ejemplos de entrenamiento con un segundo conjunto de ejemplos de validación y seleccionar aquél valor de λ que minimice el error.

Aplica esta técnica con los valores de $\lambda \in \{\ 0,\ 0.001,\ 0.003,\ 0.01,\ 0.03,\ 0.1,\ 0.3,\ 1,\ 3,\ 10\ \}$ y dibuja una gráfica con los valores de error para los ejemplos de entrenamiento y los de validación que debe ser similar a esta:



Donde se comprueba que el mejor valor de λ parece ser 3. Por último, deberías estimar el error de la hipótesis aplicándola a un tercer conjunto de ejemplos que no hayas utilizado para entrenar ni tampoco para seleccionar λ. Calculando el error sobre los datos de prueba `Xtest` (que también debes pasar a forma polinomial con potencia 8 y después normalizar con las medias y normales de los datos de entrenamiento) e `ytest`, para $\lambda = 3$ deberías obtener un error en torno a 3,572.

5. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual. Se entregará un único fichero en formato pdf que contenga la memoria de la práctica, incluyendo el código desarrollado y los comentarios y gráficas que se estimen más adecuados para explicar los resultados obtenidos.