Práctica 5: Regresión lineal regularizada: sesgo y varianza

Nuria Bango Iglesias (nubango@ucm.es)

Álvar Julián de Diego López (alvarded@ucm.es)

1. Regresión lineal regularizada

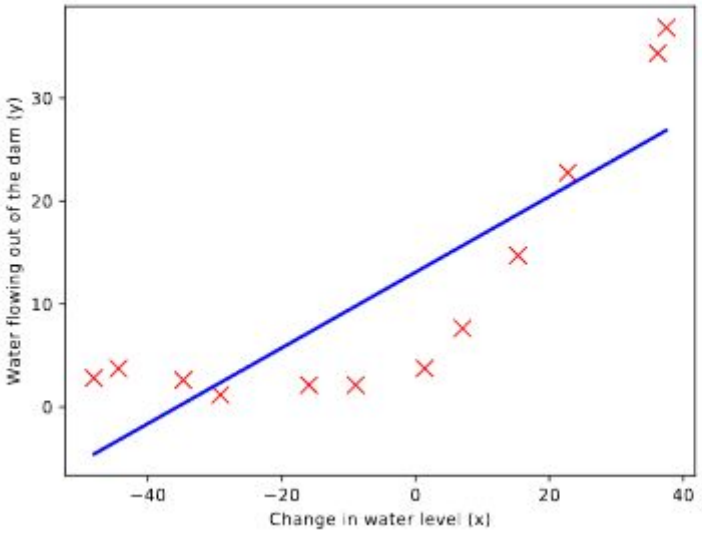
El objetivo de esta práctica es comprobar los efectos del sesgo (bias) y la varianza (variance).Aplicaremos regresión lineal regularizada para aprender una hipótesis sesgada, que no es capaz de clasificar correctamente a los ejemplos de entrenamiento, y a continuación usaremos de nuevo la regresión lineal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

```
valores = load_mat("ex5data1.mat")

X = valores['X']          # datos de entrenamiento
Y = valores['y']
Xval = valores['Xval']    # ejemplos de validacion
Yval = valores['yval']
Xtest = valores['Xtest'] # prueba
Ytest = valores['ytest']
```

A continuación vamos a implementar las funciones de coste y gradiente

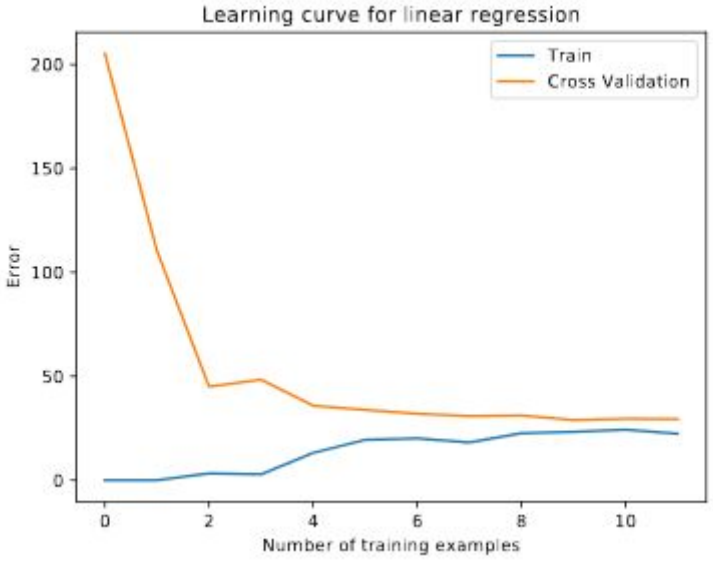
```
def coste(X, Y, Theta, reg):
    Theta = Theta[np.newaxis]
    AuxO = Theta[:, 1:]

    H = np.dot(X, Theta.T)
    Aux = (H-Y)**2
    cost = Aux.sum()/(2*len(X))

    return cost + (AuxO**2).sum()*reg/(2*X.shape[0])

def gradiente(X, Y, Theta, reg):
    AuxTheta = np.hstack([np.zeros([1]), Theta[1:,]])
    Theta = Theta[np.newaxis]
    AuxTheta = AuxTheta[np.newaxis].T

    return ((X.T.dot(np.dot(X, Theta.T)-Y))/X.shape[0] + (reg/X.shape[0])*AuxTheta)
```

Y usamos scipy.optimize.minimize para mostrar una gráfica fijando λ = 0



2. Curvas de aprendizaje

```python
def learningCurve(errorX, errorXVal):
    """muestra el grafico de la funcion h(x)"""

    x = np.linspace(0, 12, errorX.shape[0], endpoint=True)
    xVal = np.linspace(0, 12, errorXVal.shape[0], endpoint=True)

    # pintamos funcion de estimacion
    plt.plot(x, errorX)
    plt.plot(xVal, errorXVal)
    plt.savefig('curvaAprendizaje.png')
    plt.show()
```



3. Regresión polinomial

```
Xpoly = polynomize(X, 8)
Xnorm, mu, sigma = normalize(Xpoly[:, 1:])
Xnorm = np.hstack([np.ones([Xnorm.shape[0], 1]), Xnorm])

XpolyVal = polynomize(Xval, 8)
XnormVal = normalizeValues(XpolyVal[:, 1:], mu, sigma)
XnormVal = np.hstack([np.ones([XnormVal.shape[0], 1]), XnormVal])

XpolyTest = polynomize(Xtest, 8)
XnormTest = normalizeValues(XpolyTest[:, 1:], mu, sigma)
XnormTest = np.hstack([np.ones([XnormTest.shape[0], 1]), XnormTest])
```

4. Selección del parámetro λ

```
errorX = np.zeros(l.shape[0])
errorXVal = np.zeros(l.shape[0])

# errores para cada valor de lambda
for i in range(l.shape[0]):
    result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
     args = (Xnorm, Y, l[i]), method = 'TNC', jac = True, options = {'maxiter':70})
    O = result.x

    errorX[i] = coste(Xnorm, Y, O, l[i])
    errorXVal[i] = coste(XnormVal, Yval, O, l[i])

lambdaGraphic(errorX, errorXVal, l)

# lambda que hace el error minimo en los ejemplos de validacion
lambdaIndex = np.argmin(errorXVal)
print("Mejor lambda: " + str(l[lambdaIndex]))

# thetas usando la lambda que hace el error minimo (sobre ejemplos de entrenamiento)
result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
    args = (Xnorm, Y, l[lambdaIndex]), method = 'TNC', jac = True, options = {'maxiter':70})

O = result.x
```

Vamos a calcular el error sobre los datos de prueba en forma polinomial con potencia 8 y normalizadas, para lamba = 3, debería salir entorno a 3,572

```
Mejor lambda: 0.3
```