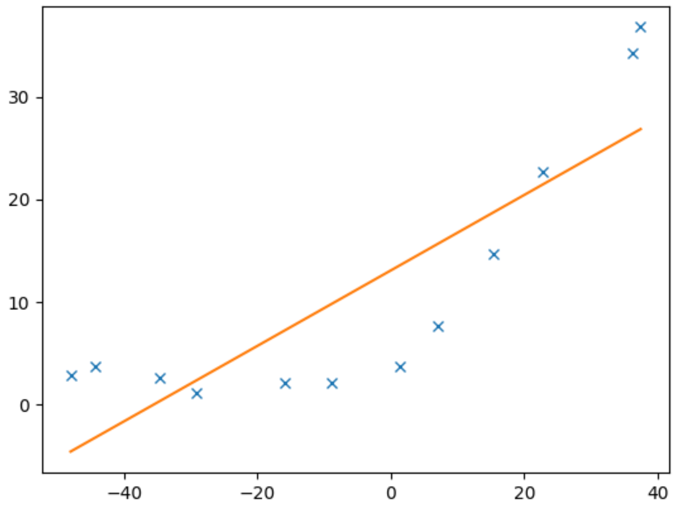
Práctica 5: Regresión lineal regularizada: sesgo y varianza

David Godoy Ruiz
Eva Lucas Leiro

Regresión lineal regularizada:

Aplicamos el método de regresión lineal regularizada. Primero calculamos el coste y el gradiente.

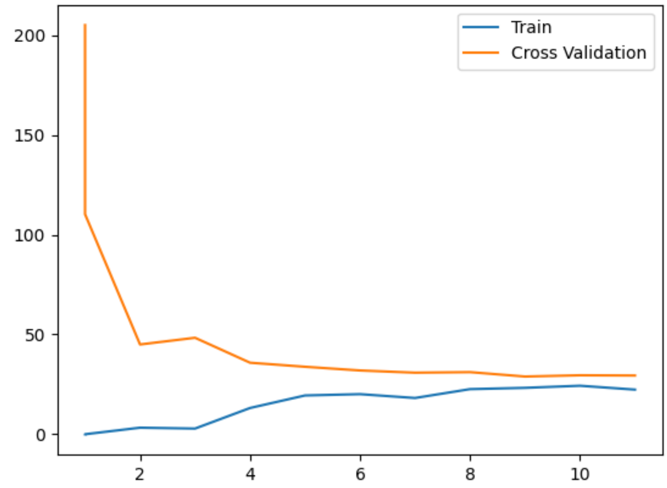
```
def coste_lineal(X, Y, Theta, lambd):
    H = np.dot(X, Theta)
    return (np.sum((H - Y.T) ** 2) + np.sum(lambd * (Theta[1:] ** 2)))/(2*len(X))

#-----------------------------------------------------------------------
def gradiente_lineal(Theta, X, Y, lambd):
    H = np.dot(X, Theta)
    gradiente = np.dot((H-Y),X)/len(X)
    gradiente[1:] = gradiente[1:] + (lambd * Theta/len(X))[1:]
    return gradiente

#-----------------------------------------------------------------------
def lineal(Theta, X, Y, lambd):
    return coste_lineal(X, Y, Theta, lambd), gradiente_lineal(Theta, X, Y, lambd)
#-----------------------------------------------------------------------
def pintaLineal(X, y):

    theta = np.array([1, 1])

    res = opt.minimize(fun=lineal,x0= theta, args= (X, y, 1), jac = True, method =
'TNC')

    plt.plot(X[:, 1], y, "x")
    min_x = min(X[:, 1])
    max_x = max(X[:, 1])
    min_y = res.x[0] + res.x[1] * min_x
    max_y = res.x[0] + res.x[1] * max_x
    plt.plot([min_x, max_x], [min_y, max_y])
    plt.savefig("resultado.png")
    plt.clf()
```



Curvas de aprendizaje:

Al ser la representación anterior demasiado simple para ajustarse a los datos de entrenamiento, utilizamos la representación mediante curvas de aprendizaje para identificar situaciones de sesgo y varianza

```
def error(X,y,reg,Xval, Yval):
    m = np.shape(X)[0]
    mV = np.shape(Xval)[0]
    errorV = np.zeros([m])
    errorE = np.zeros([m])

    Xval = np.hstack([np.ones([mV, 1]), Xval])

    for i in range(1,m+1):
        theta = np.zeros(np.shape(X)[1])
        res = opt.minimize(fun=lineal,x0= theta, args= (X[0:i], y[0:i], reg), jac =
True, method = 'TNC')
        errorV[i-1] = coste_lineal(Xval,Yval,res.x,0)
        errorE[i-1] = coste_lineal(X[0:i],y[0:i],res.x,0)

    return errorE, errorV

#-----------------------------------------------------------------------
def pintaError(errorE, errorV):

    plt.plot(np.linspace(1,11,12,dtype=int),errorE, label="Train")
    plt.plot(np.linspace(1,11,12,dtype=int),errorV, label="Cross Validation")
    plt.legend()
    plt.savefig("curvas.png")
    plt.clf()
```



Regresión polinomial:

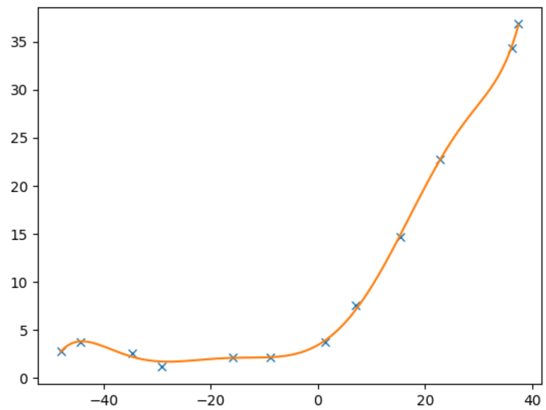
Para que el ajuste a los datos sea mayor, usamos un polinomio de x como hipótesis.

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

Primero definimos una función para generar los nuevos datos de entrenamiento a partir de los originales. Esta función devolverá una matriz m*p con una primera columna con los valores de X, una segunda con los de X^2 y así sucesivamente.

Es necesario normalizar los atributos para evitar grandes diferencias de rango.

Generamos los nuevos datos para aprender un polinomio de grado p=8 y aplicamos el método de regresión lineal de nuevo para obtener el vector θ que minimiza el error, para un valor de λ = 0. Obtenemos esta curva:

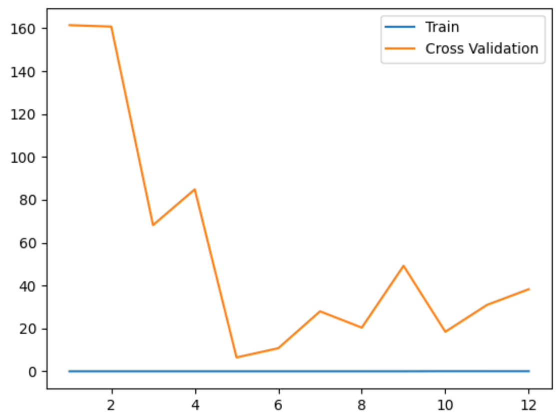


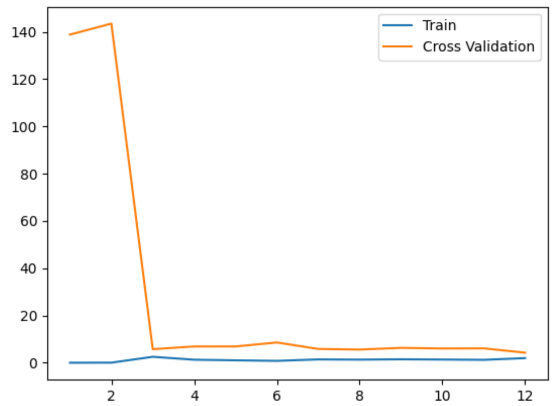
```python
def main():
    Xini, y, Xval, yval, Xtest, ytest = load()
    m = np.shape(Xini)[0]
    X = np.hstack([np.ones([m, 1]), Xini])
    y = y.ravel()

    Xexp = expandir(Xini,8)
    Xnor , mu, sigma = normalizar_mat(Xexp)
    Xnor = np.hstack([np.ones([np.shape(Xnor)[0],1]), Xnor])

    theta = np.zeros(np.shape(Xnor[1]))
    res = opt.minimize(fun=lineal,x0= theta, args= (Xnor, y, 0), jac = True,
method = 'TNC')

    pintaPolinomial(X,y,res.x,mu,sigma)

    XvalExp = expandir(Xval,8)
    XvalExp = (XvalExp-mu)/sigma

    errorPoli(Xnor, y, XvalExp, yval)
#-----------------------------------------------------------------------

def pintaPolinomial(X,y,res,mu,sigma):
    plt.plot(X[:,1],y,"x")

    lX = np.arange(np.min(X), np.max(X), 0.05)
    aux = (expandir(lX,8)-mu)/ sigma
    lY = np.hstack([np.ones([len(aux),1]),aux]).dot(res)
    plt.plot(lX,lY,'-')

    plt.savefig("polinomial.png")
    plt.clf()

#-----------------------------------------------------------------------
def errorPoli(Xnor, y, Xval, yval):
    errorE, errorV = error(Xnor,y,0,Xval, yval)
    pintaErrorPoli(errorE, errorV,'0')

    errorE, errorV = error(Xnor,y,1,Xval, yval)
    pintaErrorPoli(errorE, errorV,'1')

    errorE, errorV = error(Xnor,y,50,Xval, yval)
```

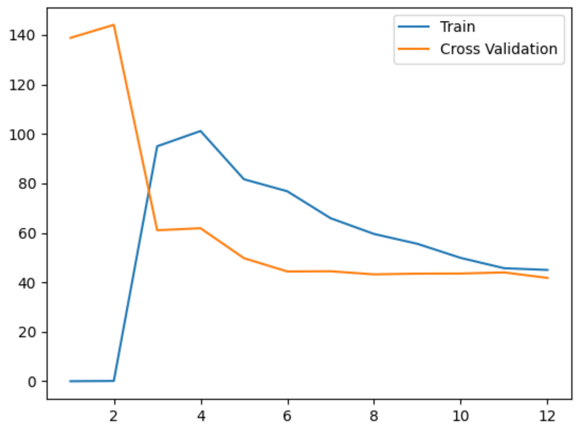
```
    pintaErrorPoli(errorE, errorV,'50')

    errorE, errorV = error(Xnor,y,100,Xval, yval)
    pintaErrorPoli(errorE, errorV,'100')
#------------------------------------------------------------------------
def pintaErrorPoli(errorE, errorV, i):
    plt.plot(np.linspace(1,len(errorE),len(errorE),dtype=int),errorE,
label="Train")
    plt.plot(np.linspace(1,len(errorV),len(errorV),dtype=int),errorV,
label="Cross Validation")
    plt.legend()
    plt.savefig("ErrorPoli"+i+".png")
    plt.clf()
```

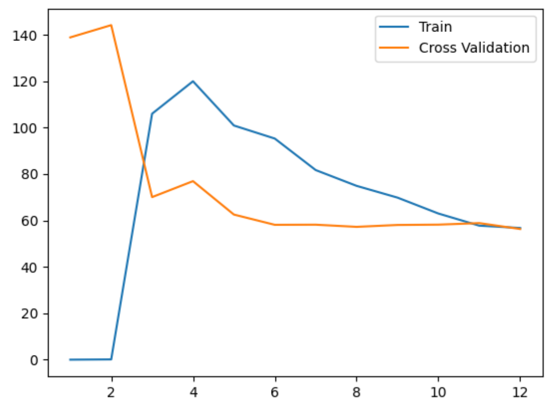
Generamos las curvas de aprendizaje para la hipótesis polinomial del mismo modo que en el apartado anterior.

Para λ = 0, 1, 50, 100 :



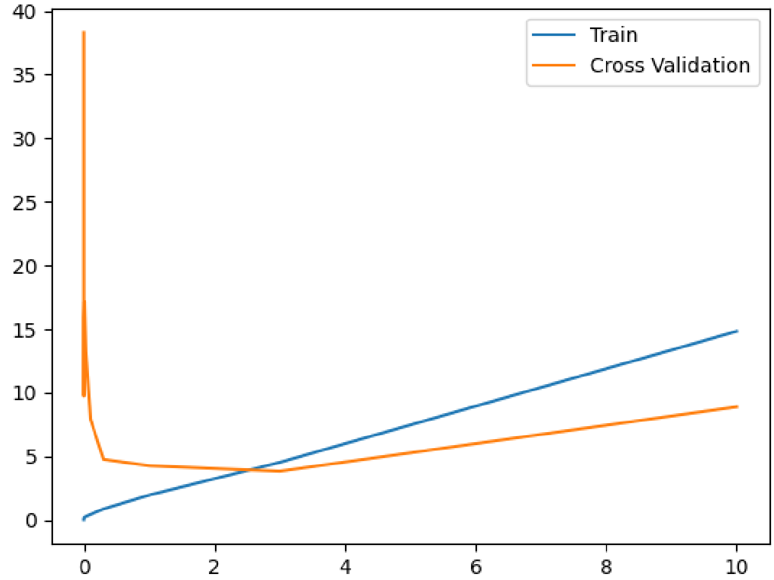






Selección del parámetro λ:

Para elegir el valor de λ evaluaremos la hipótesis generada sobre los ejemplos de entrenamiento con otro conjunto de ejemplos de validación y seleccionaremos aquel valor que minimice el error.
Al aplicarla con los valores λ ∈ { 0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10 }, obtenemos esta gráfica, donde comprobamos que el mejor valor es 3.




```
def error_lambda(X,y,reg,Xval, Yval):
    m = np.shape(X)[0]
    mV = np.shape(Xval)[0]

    Xval = np.hstack([np.ones([mV, 1]), Xval])

    theta = np.zeros(np.shape(X)[1])
    res = opt.minimize(fun=lineal,x0= theta, args= (X, y, reg), jac = True,
method = 'TNC')
    errorV = coste_lineal(Xval,Yval,res.x,0)
    errorE = coste_lineal(X,y,res.x,0)

    return errorE, errorV
#-----------------------------------------------------------------------

def errorReg(Xnor, y, Xval, yval):
    lambdas = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])
    errorE = np.empty_like(lambdas)
    errorV = np.empty_like(lambdas)
    for i in range(len(lambdas)):
```

```
        errorE[i], errorV[i] = error_lambda(Xnor,y,lambdas[i],Xval, yval)

    pintaErrorReg(errorE, errorV,lambdas)

#------------------------------------------------------------------------

def pintaErrorReg(errorE, errorV, lambdas):
    plt.clf()
    plt.plot(lambdas,errorE,label="Train")
    plt.plot(lambdas,errorV,label="Cross Validation")
    plt.legend()
    plt.savefig("ErrorLambdas")

#------------------------------------------------------------------------

def tercerConjunto(X, y, Xtest, ytest, mu, sigma):
    theta = np.zeros(np.shape(X[1]))
    res = opt.minimize(lineal, theta, args = (X, y, 3), jac = True, method =
'TNC')

    XtestExp = expandir(Xtest,8)
    XtestExp = (XtestExp-mu)/sigma
    XtestExp = np.hstack([np.ones([np.shape(XtestExp)[0],1]), XtestExp])

    error = coste_lineal(XtestExp,ytest,res.x,0)
    print(error)
```

Por último estimamos el error sobre los terceros datos para lambda 3 y obtenemos un error entorno al 3,572.