Práctica 5: Regresión lineal regularizada: sesgo y varianza

Manuel Hernández Nájera-Alesón
Mario Jiménez Contreras
Grupo 17

Parte 1: Regresión lineal regularizada

1.1 Planteamiento

Esta primera parte era la de preparar las funciones de regresión lineal, y cargar los datos de manera adecuada.

1.2 Código:

Función hipótesis:

```
def hipotesis(theta,X): #hipotesis funcion lineal
    return X.dot(theta)
```

Función de coste lineal (regularizada)

```
def coste(thetas, matrizX, vectorY, _lambda=0.):
    nMuestras = matrizX.shape[0]
    hipo = hipotesis(thetas,matrizX).reshape((nMuestras,1))
    cost = float((1./(2*nMuestras)) *
np.dot((hipo-vectorY).T,(hipo-vectorY))) +
(float(_lambda)/(2*nMuestras)) * float(thetas[1:].T.dot(thetas[1:]))
    return cost
```

Función descenso de gradiente (regularizada)

```
def gradiente(thetas, matrizX, vectorY, _lambda=0.):
    thetas = thetas.reshape((thetas.shape[0],1))
    nMuestras = matrizX.shape[0]
    grad =
(1./float(nMuestras))*matrizX.T.dot(hipotesis(thetas,matrizX)-vectorY
) + (float(_lambda)/nMuestras)*thetas
    return grad
```

Versión simplificada para la optimización de theta

```
def gradiente_min(thetas, matrizX, vectorY, _lambda=0.):
   return gradiente(thetas, matrizX, vectorY, _lambda=0.).flatten()
```

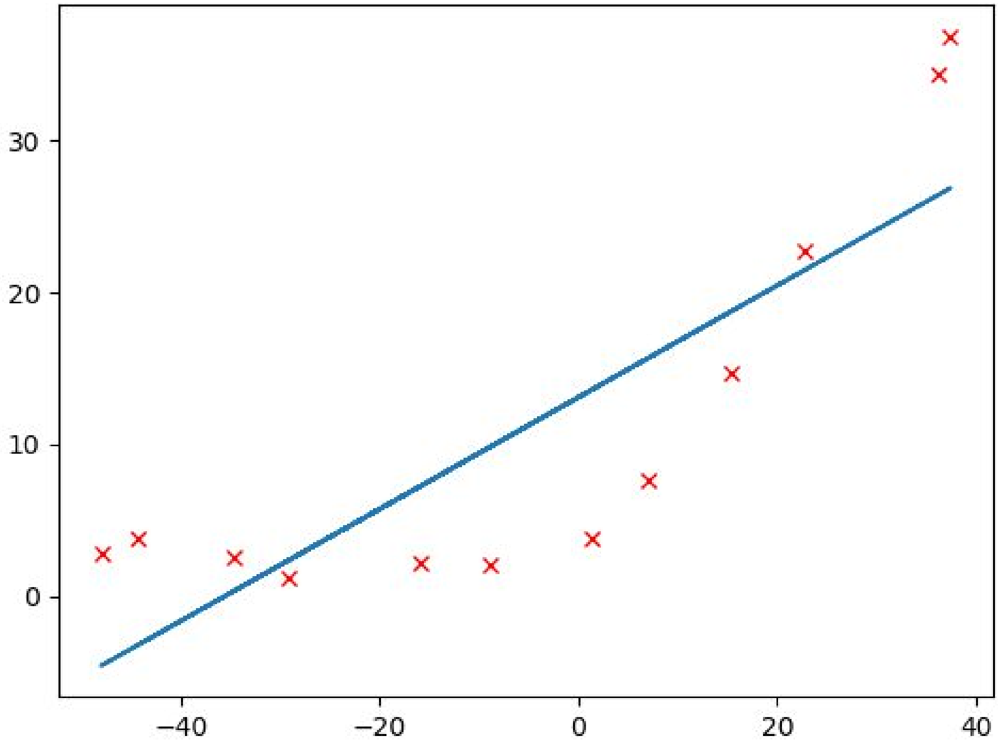
Optimización de theta

```
def optimizarTheta(thetas, matrizX, vectorY,
_lambda=0.,print_output=True):
   return opt.fmin_cg(coste,x0=thetas,fprime=gradiente_min,
args=(matrizX,vectorY,_lambda),\
                                      disp=print_output,
epsilon=1.49e-12, maxiter=1000)
```

1.3 Procedimiento:

```
#Obtención de datos
data = loadmat('ex5data1.mat')
y = data['y']
X = data['X']

#Data set
Xval = data['Xval']
yval = data['yval']

#Test set
Xtest = data['Xtest']
ytest = data['ytest']

#Columna de unos
X_unos = np.insert(X, 0, 1, axis=1)
Xval = np.insert(Xval, 0, 1, axis=1)
Xtest = np.insert(Xtest, 0, 1, axis=1)

#Apartado
1---------------------------------------------------------------------
---------------
#Grafica puntos de entrenamiento
plt.figure()
plt.plot(X, y,'rx')
#plt.show()

#Theta optimizada
Theta = np.ones((2,1))
theta_opt = optimizarTheta(Theta, X_unos, y, 0.)
plt.plot(X ,hipotesis(theta_opt, X_unos).flatten())
plt.show()
```

1.4 Resultados:



Como podemos ver, el modelo lineal no se ajusta a la muestra de entrenamiento.

Parte 2: Curvas de aprendizaje:

2.1 Planteamiento:

Utilizaremos curvas de aprendizaje para localizar el posible sobreajuste o subajuste. Para ello, utilizaremos la hipótesis de regresión lineal en diferentes subconjuntos de la muestra.

2.2 Código:

Función curva de aprendizaje con subconjuntos.

```
def curva_aprendizaje():
    theta = np.ones((2,1))
    muestras, trainVector, valVector = [], [], []
    for x in range(1,13,1):
        train_aux = X_unos[:x,:] #vamos seleccionando primer conjunto
entrenamiento, luego segundo...
        y_aux = y[:x]
        fit_theta =
optimizarTheta(theta,train_aux,y_aux,_lambda=0.,print_output=False)

trainVector.append(coste(fit_theta,train_aux,y_aux,_lambda=0.))
        valVector.append(coste(fit_theta, Xval, yval,_lambda=0.))
        muestras.append(y_aux.shape[0])

    return trainVector, valVector, muestras
```

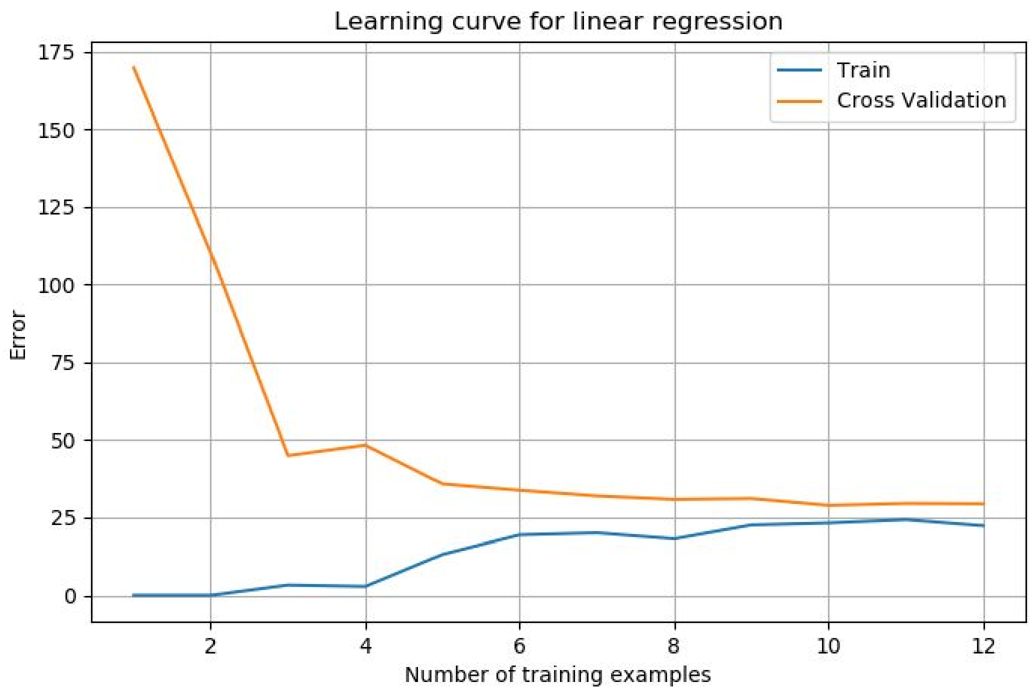
Posteriormente utilizamos esta función para pintar los resultados obtenidos:

```
def dibuja_curva_aprendizaje(vector_train, vector_val, nMuestras):
    plt.figure()
    plt.plot(nMuestras, vector_train, label='Train')
    plt.plot(nMuestras, vector_val, label='Cross Validation')
    plt.legend()
    plt.title('Learning curve for linear regression')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.show()
```

2.3 Procedimiento:

```
train_vec, val_vec, cont = curva_aprendizaje()
dibuja_curva_aprendizaje(train_vec, val_vec, cont)
```

2.4 Resultados:



Que la curva se aproxime según aumentamos el número de casos de entrenamiento indica que está sesgado.

Parte 3: Regresión polinomial:

3.1 Planteamiento:

Para poder resolver el problema anterior, vamos a cambiar la hipótesis por un polinomio de grado variable por código, para que se adapte mejor a los casos de entrenamiento.

3.2 Código:

Función que recibe una matriz X (m x 1) y devuelve una (m x p)

```
def generar_dimension(matrizX, p):
    for i in range(p):
        dim = i+2
        matrizX =
np.insert(matrizX,matrizX.shape[1],np.power(matrizX[:,1],dim),axis=1)
    return matrizX
```

Función para evitar grandes diferencias de rango

```
def normalizar_atributos(matrizX):
    medias = np.mean(matrizX,axis=0)
    matrizX[:,1:] = matrizX[:,1:] - medias[1:]
    desviaciones = np.std(matrizX,axis=0,ddof=1)
    matrizX[:,1:] = matrizX[:,1:] / desviaciones[1:]
    return matrizX, medias, desviaciones #MatrizX esta normalizada
```

Hipótesis polinomial:

```
def hipotesis_polinomial(thetas, medias, desviaciones):
    puntos = 50
    xvals = np.linspace(-55,55,puntos)
    xmat = np.ones((puntos,1))

    xmat = np.insert(xmat, xmat.shape[1],xvals.T,axis=1)
    xmat = generar_dimension(xmat,len(thetas)-2)

    xmat[:,1:] = xmat[:,1:] - medias[1:]
    xmat[:,1:] = xmat[:,1:] / desviaciones[1:]

    plt.figure()
    plt.plot(X, y,'rx')
    plt.plot(xvals,hipotesis(thetas, xmat),'b--')
    plt.show()
```

Dibujado y cálculo de curvas de aprendizaje, muy similar a la versión lineal.

```
def dibuja_curva_aprendizaje_polinomio(_lamb=0.):
    thetas = np.ones((grado_polinomio+2, 1))
    muestras, vector_train, vector_val = [], [], []
    matrizXval, aux1, aux2 =
normalizar_atributos(generar_dimension(Xval, grado_polinomio))

    for x in range(1,13,1):
        train_aux = X_unos[:x,:]
        y_aux = y[:x]
        muestras.append(y_aux.shape[0])
        train_aux = generar_dimension(train_aux, grado_polinomio)
        train_aux, aux1, aux2 = normalizar_atributos(train_aux)
        theta_opt = optimizarTheta(thetas, train_aux,
```

```
y_aux,_lambda=_lamb, _print=False)
       vector_train.append(coste(theta_opt, train_aux,
y_aux,_lambda=_lamb))
       vector_val.append(coste(theta_opt, matrizXval, yval,
_lambda=_lamb))

   plt.figure()
   plt.plot(muestras, vector_train, label='Train')
   plt.plot(muestras, vector_val, label='Cross Validation')
   plt.legend()
   plt.title('Polynomial Regression Learning Curve (lambda = ' +
str(_lamb) + ')')
   plt.xlabel('Number of training examples')
   plt.ylabel('Error')
   plt.ylim([0,100])
   plt.show()
```

3.3 Procedimiento:

Dibujado de hipótesis polinomial

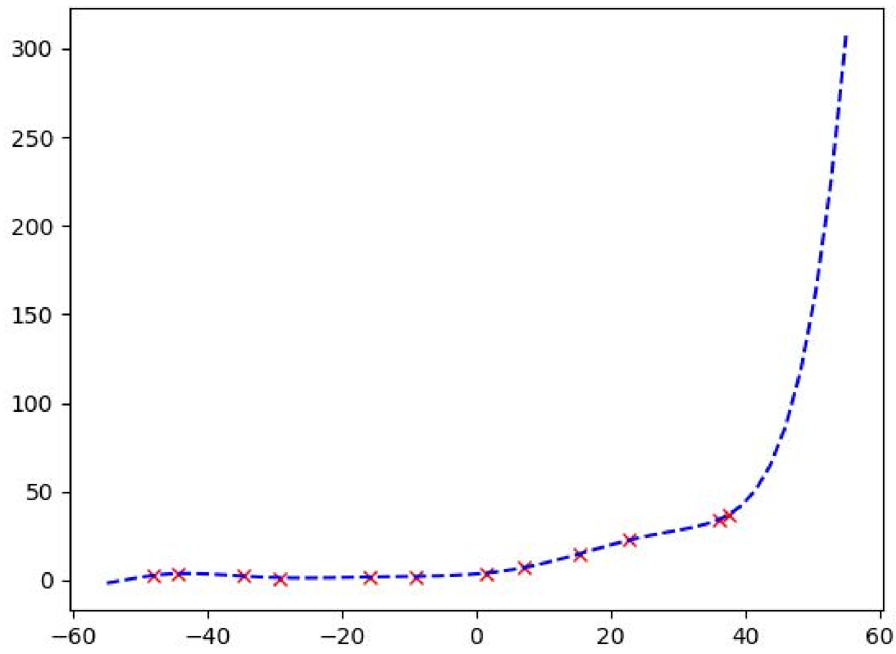
```
grado_polinomio = 8
X_poli = generar_dimension(X_unos,grado_polinomio)
X_poli_norm, media, desviacion = normalizar_atributos(X_poli)

Theta = np.ones((X_poli_norm.shape[1],1))
theta_opt = optimizarTheta(Theta, X_poli_norm, y, 0.)

hipotesis_polinomial(theta_opt, media, desviacion)
```

Dibujado curvas de aprendizaje polinómicas

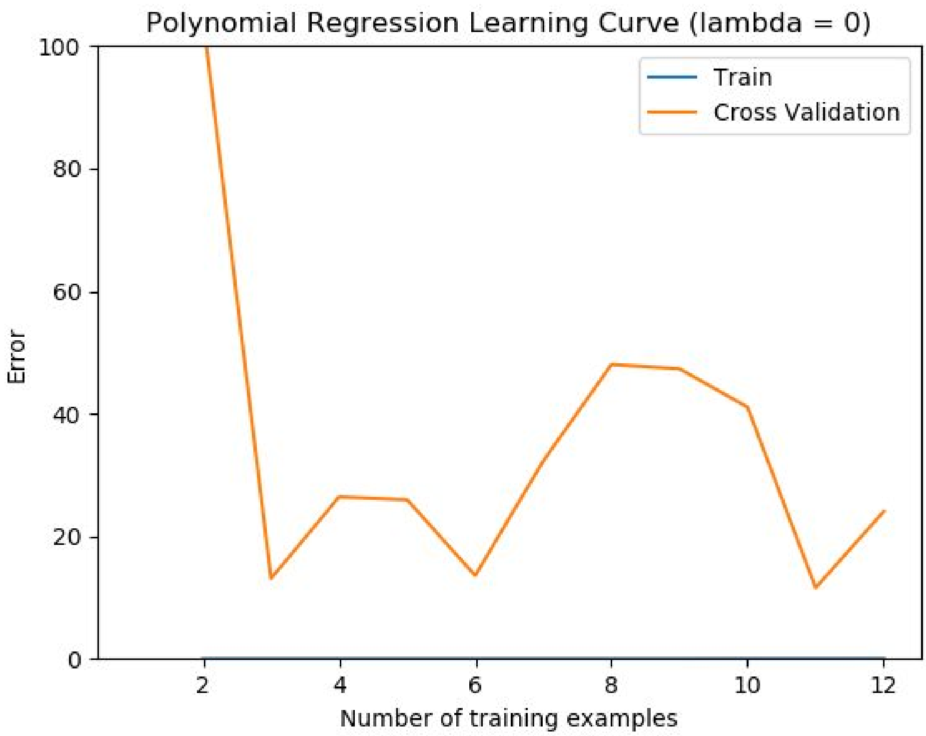
```
dibuja_curva_aprendizaje_polinomio()
```

3.4 Resultados:

La nueva hipótesis da como resultado una curva que se ajusta mejor a los casos.



Las curvas de aprendizaje:



Parte 4: Selección de Lambda:

4.1 Planteamiento:

Utilizar diferentes valores de lambda y generar la gráfica de error.

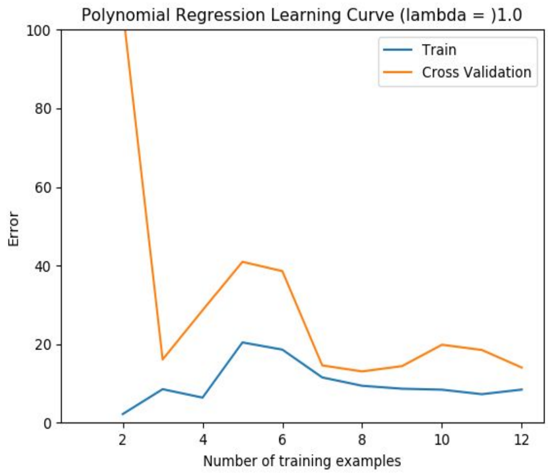
4.2 Código:

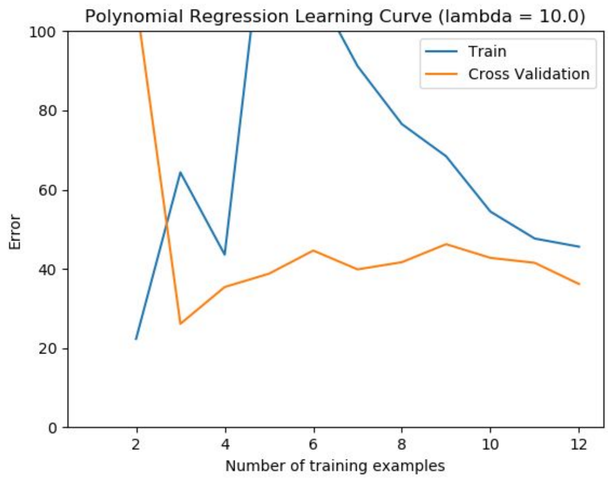
Probar con un vector de lambdas distintos y almacenar el resultado en vectores, con estos, dibujar la gráfica correspondiente.

```
Theta = np.zeros((X_poli_norm.shape[1],1))
#lambdas = [0., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1., 3., 10.]
lambdas = np.linspace(0,5,20)
vector_train, vector_val = [], []
for lamb in lambdas:
   train_aux = generar_dimension(X_unos, grado_polinomio)
   train_aux_norm, aux1, aux2 = normalizar_atributos(train_aux)
   val_aux = generar_dimension(Xval, grado_polinomio)
   val_aux_norm, aux1, aux2 = normalizar_atributos(val_aux)
   ini_theta = np.ones((X_poli_norm.shape[1],1))
   theta_opt = optimizarTheta(Theta, train_aux_norm, y, lamb, False)
   vector_train.append(coste(theta_opt, train_aux_norm, y,
_lambda=lamb))
   vector_val.append(coste(theta_opt, val_aux_norm, yval,
_lambda=lamb))

plt.figure()
plt.plot(lambdas, vector_train, label='Train')
plt.plot(lambdas, vector_val, label='Cross Validation')
plt.legend()
plt.xlabel('lambda')
plt.ylabel('Error')
plt.show()
```

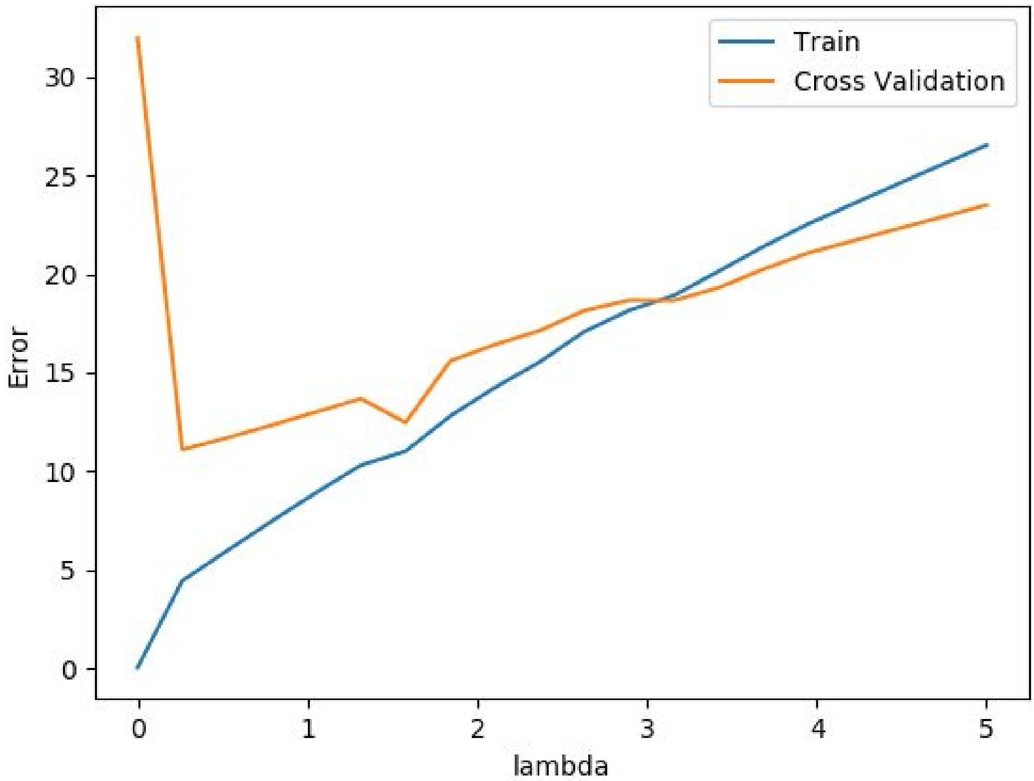
4.3 Resultados:

Curvas de aprendizaje con lambda = 1 y lambda = 10





Gráfica selección de lambda



El punto donde cruzan es el valor lambda = 3