Aprendizaje Automático y Big Data, Práctica 5

Romain Contrain y Guillermo Martín Sánchez

Mayo 2020

1 Objetivo

El objetivo de esta práctica es estudiar el sesgo (*bias*) y la varianza (*variance*) en los modelos de aprendizaje automático. En concreto estudiamos el caso de la regresión lineal regularizada con coste:

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

y gradiente:

$$\begin{cases} \frac{\delta J(\theta)}{\delta\theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} & j = 0 \\ \frac{\delta J(\theta)}{\delta\theta_j} = (\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}\theta_j & j \geq 1 \end{cases} \tag{2}$$

2 Método y Resultados

Primero hemos implementado el modelo de regresión lineal regularizada usando las ecuaciones Eq 1 y Eq 2. Hemos usado después la función *minimize* con método *TNC* de la librería *scipy.optimize* para calcular el mínimo de la función de coste regularizada (Eq 1) para los datos de entrenamiento.

Inicializando $\theta = (1, 1)$ y no usando regularización ($\lambda = 0$) obtenemos la Figura 1.

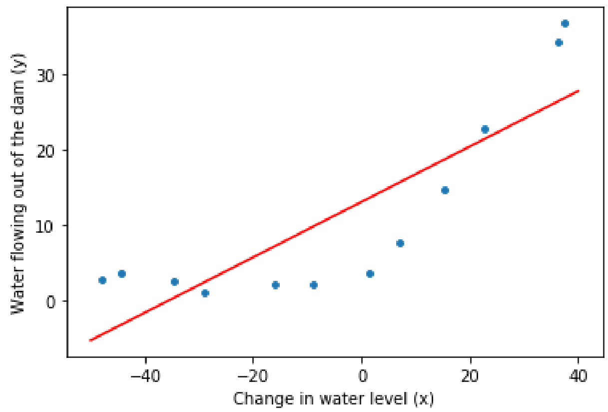

Figure 1: Modelo de regresión lineal con $\lambda = 0$

Para estudiar el sesgo y varianza de este modelo lo vamos entrenando para cada vez más elementos del dataset de entrenamiento: en el paso i lo entrenamos con los i primeros ejemplos. En cada paso calculamos el error del modelo entrenado para dichos i ejemplos de entrenamiento y para todos los ejemplos del dataset de validación. Recordamos que el error del modelo se calcula como la función de coste sin regularización:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{3}$$

En la Figura 2 vemos las curvas de aprendizaje que han resultado. Como se puede ver la diferencia conforme aumentamos los valores entre el error de entrenamiento y el de validación se hace cada vez más pequeña, pero ambos convergen a un valor alto de error, lo que nos está indicando que nuestro modelo sufre de un alto sesgo (*high bias*) o, equivalentemente, *underfitting*.

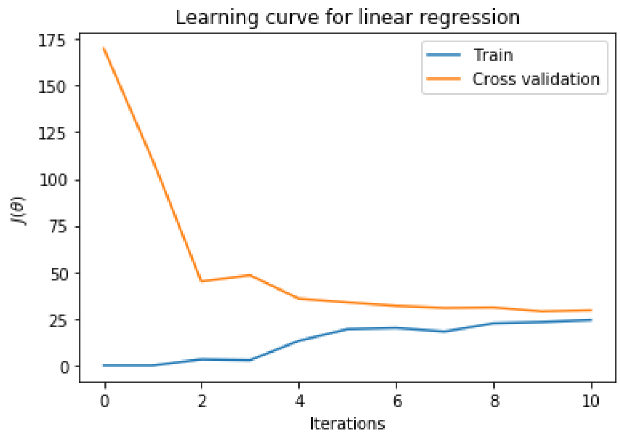

Figure 2: Curvas de aprendizaje del modelo de regresión lineal con $\lambda = 0$

Para intentar remediar esto, vamos a aumentar el número de dimensiones haciendo una transformación polinómica de los datos. Por ello, combinamos los atributos originales para extender cada punto con términos polinómicos de x hasta la octava potencia (es decir $x \longrightarrow (1, x, x^2 ... x^8)$). Utilizamos para ello *PolynomialFeatures* de la librería *sklearn.preprocessing*. Además, como ahora estos atributos tienen muy distintas escalas, los normalizamos.

Sin regularización obtenemos el modelo de la Figura 3. Después, repetimos el cálculo de las curvas de aprendizaje.

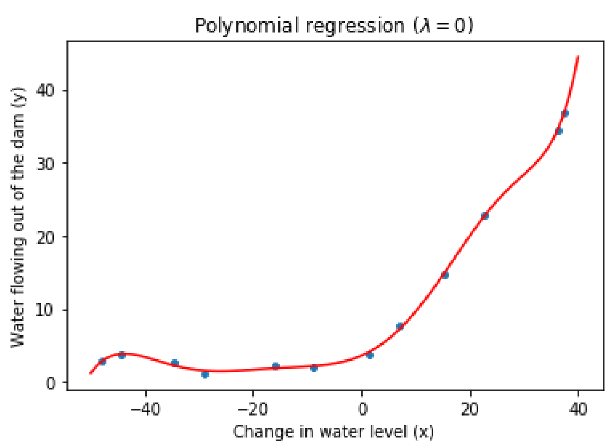

Figure 3: Modelo de regresión lineal con $\lambda = 0$

En la Figura 4 vemos el nuevo resultado aún sin regularización. En este caso no sufrimos más de *underfitting* sino todo lo contrario. La gran separación entre los errores de entrenamiento y de validación nos indica que estamos sufriendo de alta varianza (*high variance*) o, equivalentemente, *overfitting*. Una forma de evitar el *overfitting* es aumentando el valor de λ. Como se puede ver en la Figura 5 con $\lambda = 1$, obtenemos un buen modelo que no sufre ni una cosa ni de otra (ya que los errores de validación y de entrenamiento convergen a un mismo valor que es bajo). Si ponemos λ demasiado grande, como por ejemplo en la Figura 6 con $\lambda = 100$, volvemos a tener un problema de *underfitting*.

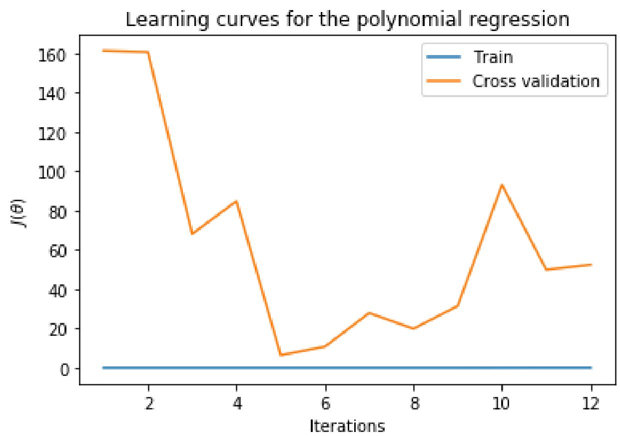

Figure 4: Curvas de aprendizaje del modelo de regresión polinómica con $\lambda = 0$

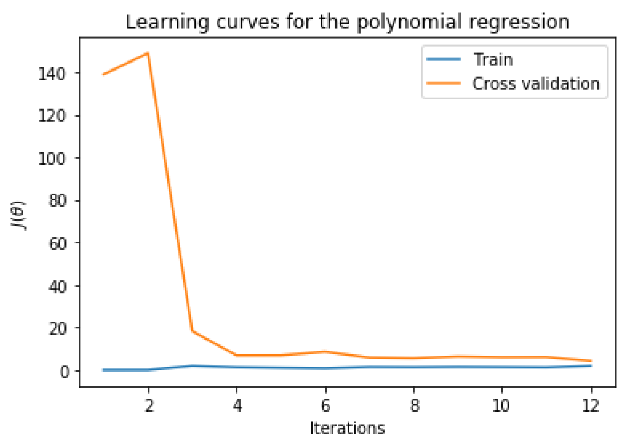

Figure 5: Curvas de aprendizaje del modelo de regresión polinómica con $\lambda = 1$

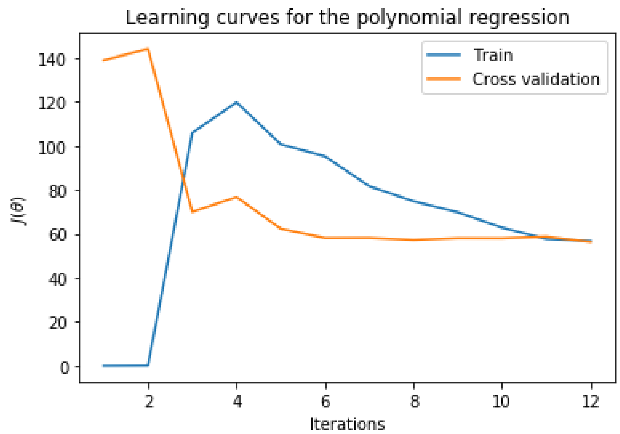

Figure 6: Curvas de aprendizaje del modelo de regresión polinómica con $\lambda = 100$

Por lo tanto, tenemos que calcular el valor adecuado de λ. Para ello dibujamos el error del modelo entrenado con todos los ejemplos de entrenamiento sobre los datos de validación, para distintos valores de λ y buscamos el λ que minimice el error de validación.

En la Figura 7 vemos dicha gráfica y vemos que el mejor λ es $\lambda = 3$.

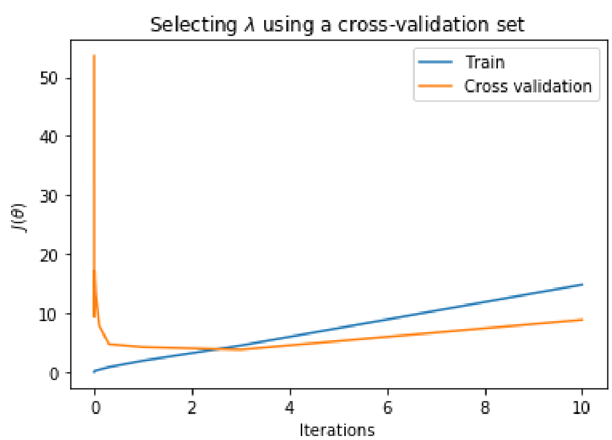

Figure 7: Error del dataset de validación para distintos valores de λ

Finalmente, con todo esto considerado, tenemos que calcular el error sobre los datos de test del mejor modelo que hemos encontrado durante la práctica (atributos polinomiales con $\lambda = 3$) y obtenemos un error bajo, del 3.572%

3 Conclusiones

El estudio de las curvas de aprendizaje del dataset de validación y el dataset de entrenamiento, nos da información útil de si estamos sufriendo de alto sesgo o alta varianza. Además, un método para solventar el alto sesgo es aumentar los atributos polinomialmente, y para solventar la alta varianza es aumentar el valor del parámetro de regularización λ. Sin embargo, para no aumentarlo demasiado es adecuado hacer la gráfica de error del dataset de validación con respecto a λ para buscar el valor que lo hace mínimo. Finalmente, el estudio final de cómo de eficiente es nuestro modelo se debe hacer calculando el error sobre un dataset nuevo: el dataset de test.

4 Código

```
# -*- coding: utf-8 -*-
"""
# Coursework 5: Regularized linear regression : bias and variance
"""

#%%
"""
Imports and definitions
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.preprocessing import PolynomialFeatures
import scipy.optimize as opt


def load_dat(file_name):
    """
    carga el fichero dat (Matlab) especificado y lo
    devuelve en un array de numpy
    """
    data = loadmat(file_name)
    y = data['y']
    X = data['X']
    ytest = data['ytest']
    Xtest = data['Xtest']
    yval = data['yval']
    Xval = data['Xval']
    return X,y,Xtest,ytest,Xval,yval
```

```python
def costgrad(theta, X, Y, lamb):
    """
    Computes the cost function and its gradient
    for a linear regression parametrized by theta
    on a dataset described by X and Y
    and a regularization parameter lamb
    """
    grad = gradiente(X, Y, theta, lamb)
    cost = coste(X, Y, theta, lamb)
    return cost,grad

def gradiente(X, Y, theta, lamb):
    """
    Computes the gradient of the cost function
    for a linear regression parametrized by theta
    on a dataset described by X and Y
    and a regularization parameter lamb
    """
    G = np.zeros((np.shape(X)[1],1))
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    H = np.dot(X,theta)
    G = 1/m * X.transpose().dot(H - Y)
    reg = np.ones(n,)
    reg[0] = 0
    reg = (lamb/m)*reg*theta
    return G + reg

def coste(X, Y, theta, lamb):
    """
    Computes the cost function
    for a linear regression parametrized by theta
    on a dataset described by X and Y
    and a regularization parameter lamb
    """
    H = np.dot(X, theta)
    Aux = (H - Y)**2
    J = Aux.sum() / (2*len(X))
    reg = lamb/(2*m)*np.sum(theta[1:]**2)
    return J + reg

def normalizar(X):
    """
    Normalizes the dataset vector X
    and returns the normalized dataset
    along with the means and the standard deviantion
```

```
    for each feature
    """
    mu = np.mean(X,  axis=0)
    sigma = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return X_norm, mu, sigma

def polynomial(X,grad):
    """
    Computes each power of X's values from 0 to grad
    and then normalizes them.
    """
    poly = PolynomialFeatures(grad)
    X_pol = poly.fit_transform(X)
    X_pol[:,1:], mu, sigma = normalizar(X_pol[:,1:])
    return X_pol, mu, sigma

#%%
"""
part 1 : regularized linear regression
"""

X,y,_,_,_,_ = load_dat('ex5data1.mat')

m = np.shape(X)[0]
# Add the column of 1s
X = np.hstack([np.ones([m,1]),X])
y = np.reshape(y,-1)

# Train the model
theta0 = np.array([[1],[1]])
lamb = 0
maxIter = 100
trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                        args=(X, y, lamb),
                        method='TNC', jac=True,
                        options={'maxiter': maxIter})
theta_opt = trainingresult.x

# Plot the data and the model
x = np.linspace(-50,40,100)
plt.scatter(X[:,1],y,s=15)
plt.plot(x, theta_opt[0] + theta_opt[1]*x, c='r')
plt.xlabel('Change in water level (x)')
plt.ylabel('Water flowing out of the dam (y)')
plt.show()
```

```python
#%%
"""
part 2 : Learning curves
"""

X,y,_,_,Xval,yval = load_dat('ex5data1.mat')

m = np.shape(X)[0]
mval = np.shape(Xval)[0]
# Add the column of 1s
X = np.hstack([np.ones([m,1]),X])
y = np.reshape(y,-1)
Xval = np.hstack([np.ones([mval,1]),Xval])
yval = np.reshape(yval,-1)

# Train the model on various subsets of the training set
# and compute its error on the training set and on the
# validation set

theta0 = np.array([[1],[1]])
lamb = 0
maxIter = 100

costs = np.zeros(m-1,)
costsval = np.zeros(m-1,)

for i in range(1,m):
    Xi = X[0:i]
    yi = y[0:i]

    trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                        args=(Xi, yi, lamb),
                        method='TNC', jac=True,
                        options={'maxiter': maxIter})
    theta_opt = trainingresult.x

    costs[i-1] = coste(Xi,yi,theta_opt,lamb)
    costsval[i-1] = coste(Xval,yval,theta_opt,lamb)

# Display the learning curves
plt.figure()
plt.plot(costs, label = "Train")
plt.plot(costsval, label = "Cross validation")
plt.legend()
```

```
plt.xlabel('Iterations')
plt.ylabel(r'$J(\theta)$')
plt.title('Learning curve for linear regression')
plt.show()


#%%
"""
part 3 : polynomial regression
"""

X,y,_,_,_,_ = load_dat('ex5data1.mat')

X_pol, mu, sigma = polynomial(X,8) #adding polynomial features
y = np.reshape(y,-1)

# Train the model
theta0 = np.zeros((9,1))
lamb = 0
maxIter = 1000

trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                              args=(X_pol, y, lamb),
                              method='TNC', jac=True,
                              options={'maxiter': maxIter})
theta_opt = trainingresult.x

# Plot the data and the polynomial
x = np.linspace(-50,40,100)
x = np.reshape(x,(-1,1))
poly = PolynomialFeatures(8)
x_pol = poly.fit_transform(x)
x_pol[:,1:] = (x_pol[:,1:] - mu) / sigma #normalizing the polynomial
                                          #the way x was normalized
plt.figure()
plt.scatter(X,y,s=15)
plt.plot(x, theta_opt.dot(x_pol.T), c='r')
plt.xlabel('Change in water level (x)')
plt.ylabel('Water flowing out of the dam (y)')
plt.title(r'Polynomial regression ($\lambda = 0 $)')
plt.show()


#%%
"""
Computing the learning curve for the polynomial regression
```

```
"""

X,y,_,_,Xval,yval = load_dat('ex5data1.mat')
m = np.shape(X)[0]

# Add the column of 1s
X_pol, mu, sigma = polynomial(X,8)
y = np.reshape(y,-1)

poly = PolynomialFeatures(8)
Xval_pol = poly.fit_transform(Xval)
Xval_pol[:,1:] = (Xval_pol[:,1:] - mu) / sigma
yval = np.reshape(yval,-1)

# Train the model on various subsets of the training set
# and compute its error on the training set and on the
# validation set

theta0 = np.zeros((9,1))
lamb = 0
maxIter = 1000

costs = np.zeros(m,)
costsval = np.zeros(m,)

for i in range(1,m+1):
    Xi = X_pol[0:i]
    yi = y[0:i]


    trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                            args=(Xi, yi, lamb),
                            method='TNC', jac=True,
                            options={'maxiter': maxIter})
    theta_opt = trainingresult.x
    costs[i-1] = coste(Xi,yi,theta_opt,0)
    costsval[i-1] = coste(Xval_pol,yval,theta_opt,0)

# Display the learning curves
plt.figure()
plt.plot(range(1,m+1), costs, label = "Train")
plt.plot(range(1,m+1), costsval, label = "Cross validation")
plt.legend()
plt.xlabel('Iterations')
plt.ylabel(r'$J(\theta)$')
plt.title('Learning curves for the polynomial regression')
```

```python
plt.show()


#%%
"""
part 4 : selecting parameter lambda
"""

X,y,_,_,Xval,yval = load_dat('ex5data1.mat')
m = np.shape(X)[0]

# Add the column of 1s
X_pol, mu, sigma = polynomial(X,8)
y = np.reshape(y,-1)

poly = PolynomialFeatures(8)
Xval_pol = poly.fit_transform(Xval)
Xval_pol[:,1:] = (Xval_pol[:,1:] - mu) / sigma
yval = np.reshape(yval,-1)

# Train the model for a range of values of lambda
# and compute its error on the training set and on the
# validation set

theta0 = np.ones((9,1))
lamb_arr = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
maxIter = 1000

costs = np.zeros(10,)
costsval = np.zeros(10,)

i = 0
for lamb in lamb_arr:
    trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                                  args=(X_pol, y, lamb),
                                  method='TNC', jac=True,
                                  options={'maxiter': maxIter})
    theta_opt = trainingresult.x
    costs[i] = coste(X_pol,y,theta_opt,0)
    costsval[i] = coste(Xval_pol,yval,theta_opt,0)
    i = i + 1

# Display the learning curves
plt.figure()
plt.plot(lamb_arr, costs, label = "Train")
plt.plot(lamb_arr, costsval, label = "Cross validation")
```

```python
plt.legend()
plt.xlabel('Iterations')
plt.ylabel(r'$J(\theta)$')
plt.title(r'Selecting $\lambda$ using a cross-validation set')
plt.show()


#%%
"""
Computing the error on the test set with the ideal lambda
"""

X,y,Xtest,ytest,Xval,yval = load_dat('ex5data1.mat')
m = np.shape(X)[0]

# Add the column of 1s
X_pol, mu, sigma = polynomial(X,8)
y = np.reshape(y,-1)

poly = PolynomialFeatures(8)
Xtest_pol = poly.fit_transform(Xtest)
Xtest_pol[:,1:] = (Xtest_pol[:,1:] - mu) / sigma
ytest = np.reshape(ytest,-1)

# Train the model
theta0 = np.ones((9,1))
lamb = 3
maxIter = 1000

trainingresult = opt.minimize(fun=costgrad, x0=theta0,
                        args=(X_pol, y, lamb),
                        method='TNC', jac=True,
                        options={'maxiter': maxIter})
theta_opt = trainingresult.x

# Display the error on the test set
print("Cost obtained in test set:", coste(Xtest_pol,ytest,theta_opt,0))
```