Práctica 5

Alejandro Aizel Boto
Miguel Robledo Blanco

Índice

1. Descripción General

Esta práctica consiste en aplicar los conceptos de sesgo (subajuste) y varianza (sobreajuste) a una hipótesis demasiado sencilla como para clasificarla correctamente. Una vez se compruebe que la hipótesis está sesgada, se mostrarán las gráficas de aprendizaje.

Una vez hecho esto, se aplicará una regresión polinomial, es decir, se añadirán términos a la hipótesis para sobreajustarla a los datos. A su vez, también se mostrarán las curvas de aprendizaje.

2. Explicación de la Solución

El conjunto de datos de entrenamiento viene dividido en un diccionario proporcionado por el profesor en conjunto de entrenamiento `X`, conjunto de validación `Xval` y conjunto de prueba `Xtest`, cada uno con su respectiva `Y`;

```
def main():
    data = loadmat('Práctica 5/Recursos/ex5data1.mat')

    X, Y = data['X'], data['y'].ravel()
    X_val, Y_val = data['Xval'], data['yval'].ravel()
    X_test, Y_test = data['Xtest'], data['ytest'].ravel()
    Landa, p = 0, 8

    linear_regression(X, Y)
    learning_curve(X, Y, X_val, Y_val, Landa)
    polinomial_regression(X, Y, X_val, Y_val, Landa, p)
    parameter_selection(X, Y, X_val, Y_val, X_test, Y_test, p)
```

Definimos una función main en la que cargamos los datos y llamamos a cuatro funciones, una para cada apartado de la práctica. Vamos a comenzar con la primera función del apartado 1 `linear_regression()`.

```
def linear_regression(X, Y):
    X = np.hstack((np.ones((len(X), 1)), X))

    Theta = np.ones(2)
    Landa = 1

    cost, grad = cost_gradient_reg(Theta, X, Y, Landa)

    print("For Theta = {} and Lamda = {}:\n  Cost = {}\n  Gradient =
        {}".format(Theta, Landa, cost, grad))

    Landa = 0

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(X, Y, Landa),
        method='TNC', jac=True)

    linear_regression_graph(X[:,1:], Y, fmin.x, name="linear_regression")
```

En esta primera función se calcula la hipótesis sesgada. Añadimos una columna de unos como siempre para facilitar el cálculo, y llamamos a la función `cost_gradient_reg()` que calcula el coste y el gradiente regularizado para una hipótesis e imprimimos el resultado por pantalla. Además, llamamos a la función minimize de `scipy.optimize` que nos devuelve el vector de theta óptimo para luego llamar a `linear_regression_graph()` que nos muestra la gráfica de la función sesgada. A continuación se muestran las funciones utilizadas que no se van a explicar por estar explicadas en prácticas anteriores.

```
def gradient(Theta, X, Y):
    H = np.matmul(X, Theta)

    return np.matmul(np.transpose(X), H - Y) / len(Y)

def cost(Theta, X, Y):
    H = np.dot(X, Theta)

    aux = (H - Y) ** 2

    return aux.sum() / (2 * len(X))

def gradient_reg(Theta, X, Y, Landa):
    G = gradient(Theta, X, Y)

    G[1:] += Landa / len(X) * Theta[1:]

    return G

def cost_reg(Theta, X, Y, Landa):
    C = cost(Theta, X, Y)

    aux2= np.sum(Theta[1:] ** 2)
    aux1 = (2 * len(X))

    aux = aux1 * aux2

    return C + Landa / aux

def cost_gradient_reg(Theta, X, Y, Landa):
    return [cost_reg(Theta, X, Y, Landa), gradient_reg(Theta, X, Y, Landa)]
```

Para mostrar la gráfica tenemos el siguiente código:

```
def linear_regression_graph(X, Y, Theta, name):
    plt.figure()

    X_l = [np.amin(X), np.amax(X)]
    Y_l = [Theta[0] + Theta[1] * X_l[0], Theta[0] + Theta[1] * X_l[1]]

    plt.plot(X, Y, 'x', c='r')
    plt.plot(X_l, Y_l, c='b')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)
```

Lo que hacemos en esta función para dibujarla es, además de dibujar los puntos de X e Y, hallar los dos puntos extremos usando las Thetas con el X más pequeño y más grande para así poder dibujar la recta que una a esos dos puntos.

Seguimos con el apartado 2 llamando a la función `learning_curve()`.

```
def learning_curve(X, Y, X_val, Y_val, Landa, name="learning_curves"):
    m = len(X)

    X = np.hstack((np.ones((len(X), 1)), X))
    X_val = np.hstack((np.ones((len(X_val), 1)), X_val))

    Theta = np.ones(len(X[0]))

    cost = []
    cost_val = []

    for i in range(1, m + 1):
        fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(X[0:i],
             Y[0:i], Landa), method='TNC', jac=True)

        cost.append(fmin['fun'])
        cost_val.append(cost_gradient_reg(fmin.x, X_val, Y_val, Landa)[0])

    learning_curve_graph(cost, cost_val, "Number of training examples",
         "Learning curve for lineal regression (λ = {})".format(Landa),
              name=name)
```

En esta segunda parte tenemos que mostrar las curvas de aprendizaje entre el conjunto `X` y el conjunto de validación `X_val`, para ello las gráficas mostrarán el coste para ambos conjuntos llegando a la conclusión de que ambos costes tienen que ser similares.

Se empieza añadiendo una columna de unos a los conjuntos y se definen los arrays de coste. En el bucle for se llama a la función `minimize()` de `scipy.optimize`. Para el conjunto de `X` simplemente se añade al array `fmin['fun']` que contiene directamente el coste, para `X_val` se llama a la función `cost_gradient_reg()` con `fmin.x` que contiene el vector de Thetas y nos quedamos con el primer valor que devuelve la función (el coste).

Posteriormente llamamos a `learning_curve_graph()` para mostrar el resultado de forma gráfica.

```
def learning_curve_graph(X1, X2, x_label, title, name):
    plt.figure()

    Y = range(len(X1))

    plt.title(title)
    plt.plot(Y, X1, c='orange', label="Train")
    plt.plot(Y, X2, c='blue', label="Cross Validation")
    plt.xlabel(x_label)
    plt.ylabel("Error")
    plt.legend()
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)
```

Lo que hacemos en esta función es generar las dos gráficas con cada valor del error en cada punto generando una serie de puntos en el eje de abscisas con el mismo tamaño que el número de puntos de estas.

A continuación llamamos a la tercera función `polinomial_regression()` correspondiente al tercer apartado.

```
def polinomial_regression(X, Y, X_val, Y_val, Landa, p):
    new_X = polynomial_features(X, p)
    norm_X, mu_X, sigma_X = normalize(new_X)
    norm_X = np.hstack((np.ones((len(norm_X), 1)), norm_X))

    Theta = np.ones(len(norm_X[0]))

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y,
        Landa), method='TNC', jac=True)

    polynomial_regression_graph(X, Y, fmin.x, mu_X, sigma_X, p, "Learning
        curve for lineal regression (λ = {})".format(Landa),
            "polinomial_regression_graph")

    new_X_val = polynomial_features(X_val, p)
    norm_X_val = (new_X_val - mu_X) / sigma_X

    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=0,
        name="learning_curves_2_λ0")
    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=1,
        name="learning_curves_2_λ1")
    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=100,
        name="learning_curves_2_λ100")
```

Empezamos llamando a `polynomial_features()` para añadir términos polinómicos a `X`. Esta función añade p columnas a `X` de forma que cada vez la columna está elevada a `p`, empezando por `p=1` hasta `p=8`.

```
def polynomial_features(X, p):
    for i in range(2, p + 1):
        aux = X[:, 0] ** i

        X = np.hstack((X, np.reshape(aux, (aux.shape[0], 1))))

    return X
```

Una vez estén los datos añadidos hay que normalizarlos, calculando la media y la desviación por columnas.

```
def normalize(X):
    Mu = np.mean(X, axis=0)
    Sigma = np.std(X,axis=0)
    X_norm = (X - Mu) / Sigma

    return [X_norm, Mu, Sigma]
```

Estas dos funciones no merecen mayor explicación ya que están explicadas en prácticas anteriores.

Una vez estén normalizados, se añade la columna de unos y se llama a la función `minimize()`. Después llamamos a la función `polynomial_regression_graph()`.

```
def polynomial_regression_graph(X, Y, Theta, Mu, Sigma, p, title, name):
    plt.figure()

    X_t = np.arange(np.amin(X) - 5, np.amax(X) + 5, 0.05)
    X_t = X_t.reshape(-1, 1)
    X_p = polynomial_features(X_t, p)
    X_p = (X_p - Mu) / Sigma
    X_p = np.hstack((np.ones((len(X_p), 1)), X_p))
    Y_t = np.dot(X_p, Theta)

    plt.title(title)
    plt.plot(X, Y, 'x', c='r')
    plt.plot(X_t, Y_t, c='b')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)
```

En esta función se calculan puntos entre el máximo y el mínimo de `X` (hemos sumado y restado 5 respectivamente a cada uno para así asemejarse más al ejemplo) a intervalos de 0,05, se les añaden columnas con `polynomial_features()`, se normalizan los datos con `mu` y `sigma`, se añade la columna de unos y se calculan las `Y` en función de la `Theta` calculada anteriormente.

Por último llamamos a la última función `parameter_selection()` que hace referencia al punto 4 de la práctica.

```
def parameter_selection(X, Y, X_val, Y_val, X_test, Y_test, p):
    Landas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
    cost = []
    cost_val = []

    new_X = polynomial_features(X, p)
    norm_X, Mu, Sigma = normalize(new_X)
    norm_X = np.hstack((np.ones((len(norm_X), 1)), norm_X))
```

```
    new_X_val = polynomial_features(X_val, p)
    norm_X_val = (new_X_val - Mu) / Sigma
    norm_X_val = np.hstack((np.ones((len(norm_X_val), 1)), norm_X_val))

    Theta = np.ones(len(norm_X[0]))

    for i in Landas:
        fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y,
            i), method='TNC', jac=True)

        cost.append(fmin['fun'])
        cost_val.append(cost_reg(fmin.x, norm_X_val, Y_val, i))

    learning_curve_graph2(Landas, cost, cost_val, "Lambda", "Selecting λ
        using a cross validation set", "selecting_lambda")

    test_hypotesis(norm_X, Y, X_test, Y_test, Mu, Sigma, 3, p)
```

En esta función se estima el error de los conjuntos `X` y `X_val` para distintos valores de `lambda`. Para cada una de estas lambdas se calcula el error llamando a `minimize()`. Una vez obtenidos todos los costes llamamos a la función

```
def learning_curve_graph2(X, Y1, Y2, x_label, title, name):
    plt.figure()
    plt.title(title)
    plt.plot(X, Y1, c='orange', label="Train")
    plt.plot(X, Y2, c='blue', label="Cross Validation")
    plt.xlabel(x_label)
    plt.ylabel("Error")
    plt.legend()
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)
```

Después se llama a `test_hypothesis()` para calcular el error de la hipótesis sobre el conjunto `X_text` que hasta ahora no habíamos usado con el valor óptimo de lambda. Añadimos términos polinómicos a `X_test`, lo normalizamos y le añadimos la columna de unos.

```
def test_hypotesis(norm_X, Y, X_test, Y_test, Mu, Sigma, Landa, p):
    new_X_test = polynomial_features(X_test, p)
    norm_X_test = (new_X_test - Mu) / Sigma
    norm_X_test = np.hstack((np.ones((len(norm_X_test), 1)), norm_X_test))

    Theta = np.ones(len(norm_X[0]))

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y,
         Landa), method='TNC', jac=True)

    cost = cost_reg(fmin.x, norm_X_test, Y_test, Landa)

    print("Con λ = {}, el coste de X_text es: {}".format(Landa, cost))
```

3. Resultados

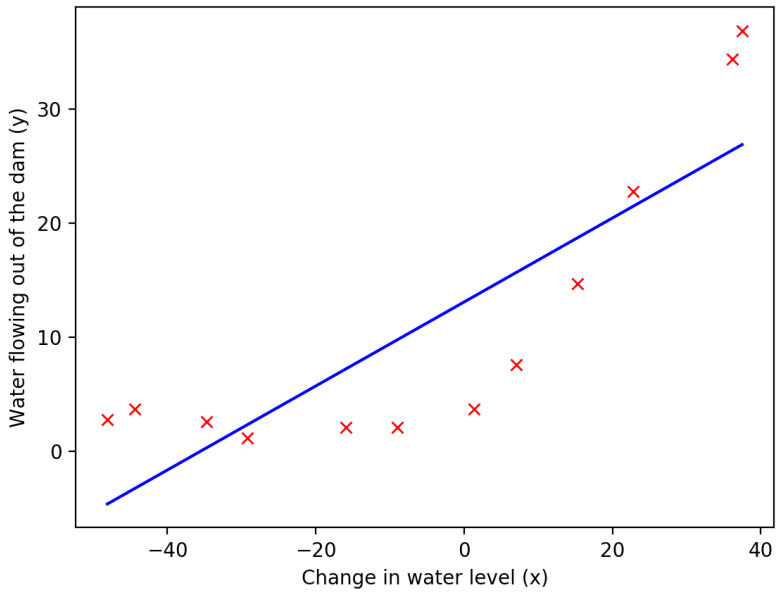
En este apartado vamos a ir comentando los distintos resultados que hemos obtenido.

Los primeros resultados que obtuvimos fueron los respectos al coste y al gradiente. En el primer apartado llamando a la función `cost_gradient_reg(Theta, X, Y, Landa)` obtenemos un resultado de coste y gradiente para un valor λ=1 y $\vartheta = [1; 1]$ que imprimimos a continuación:

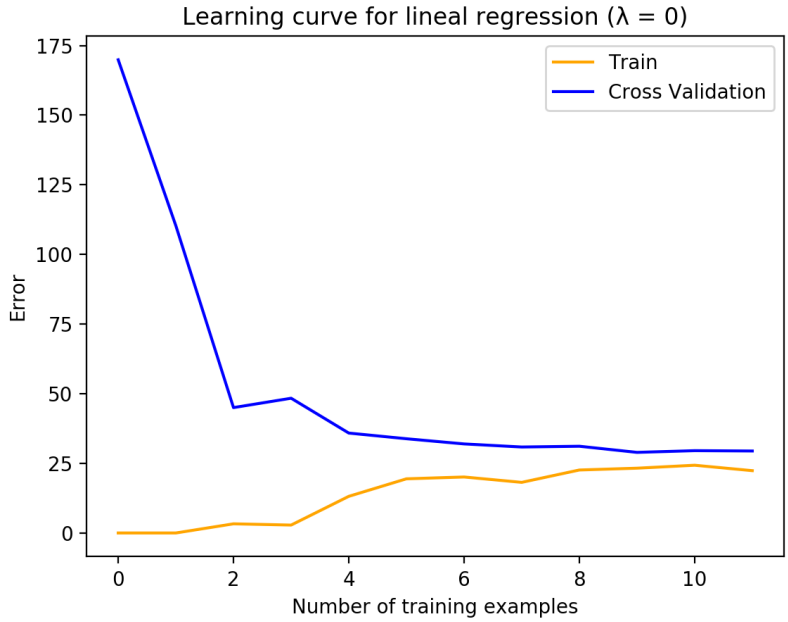
```
> print("For Theta = {} and Lamda = {}:\n  Cost = {}\n  Gradient =
      {}".format(Theta, Landa, cost, grad))
```

```
For Theta = [1. 1.] and Lamda = 1:
  Cost = 303.9931922202643
  Gradient = [-15.30301567 598.25074417]
```

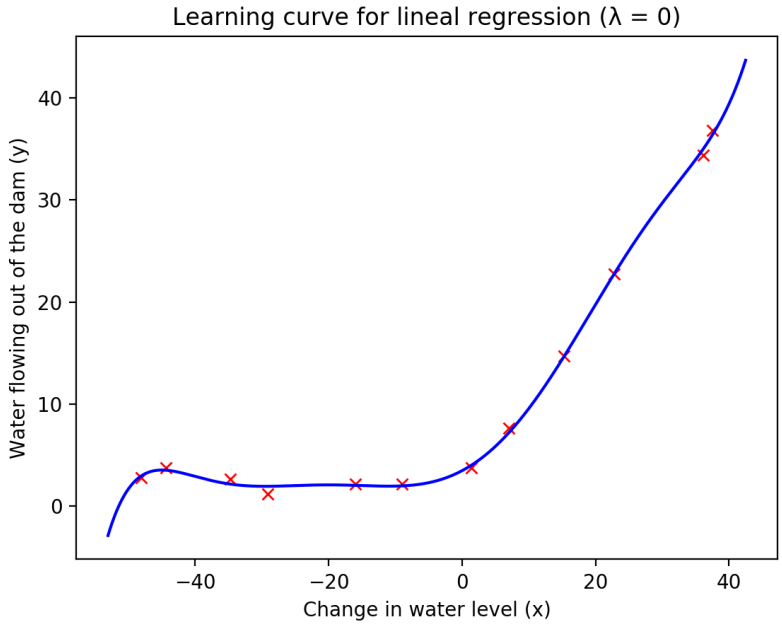
Tras llamar a la función `minimize()` la gráfica que generamos fue la siguiente:



En el segundo apartado tras llamar a la función `learning_curve_graph(cost, cost_val, "Number of training examples", "Learning curve for lineal regression (λ = {})".format(Landa), name=name)` obtenemos la siguiente gráfica:



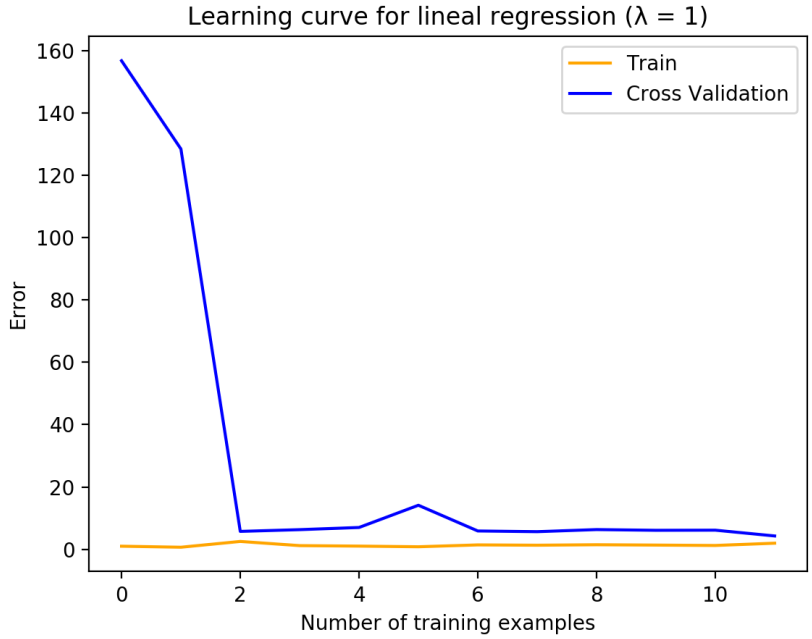
Para el tercer apartado cuando llamamos a la función `polynomial_regression_graph(X, Y, fmin.x, mu_X, sigma_X, p, "Learning curve for lineal regression (λ = {})".format(Landa), "polinomial_regression_graph")` que nos da el siguiente resultado:



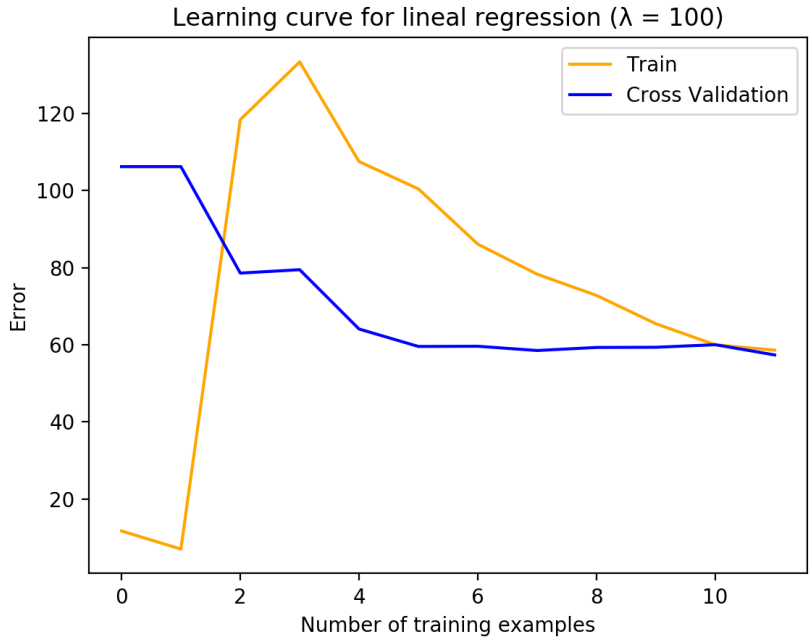
Usando la función `learning_curve()`, se muestran las curvas de aprendizaje para 3 valores distintos de Lambda (0, 1 y 100). En específico hacemos las siguientes llamadas, obteniendo cada resultado:

```
learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=0,
name="learning_curves_2_λ0")
```

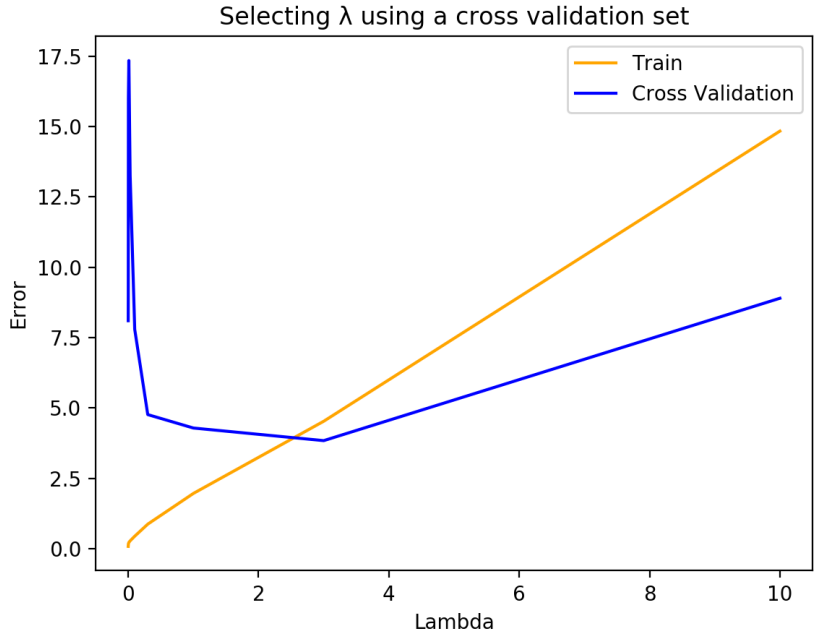

```
learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=1,
name="learning_curves_2_λ1")
```




```
learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=100,
name="learning_curves_2_λ100")
```



Para el último apartado llamamos a la función `learning_curve_graph2(Landas, cost, cost_val, "Lambda", "Selecting λ using a cross validation set", "selecting_lambda")` que nos genera la siguiente gráfica:



Finalmente la llamada a la función `test_hypotesis(norm_X, Y, X_test, Y_test, Mu, Sigma, 3, p)` genera la siguiente salida:

```
> print("With λ = {}, cost for X_text is: {}".format(Landa, cost))
```

```
  With λ = 3, cost for X_text is: 3.571583472696583
```

4. Conclusión

Esta práctica nos ha servido para comprobar los efectos del sesgo y la varianza para viaros ejemplos y ver de forma gráfica como evoluciona para así obtener el mejor valor para nuestros entrenamientos

En el siguiente apartado se muestra el código completo por si se quiere ver seguido o hacer alguna prueba compilada.

5. Código

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.io import loadmat
from scipy.optimize import minimize


def gradient(Theta, X, Y):
    H = np.matmul(X, Theta)

    return np.matmul(np.transpose(X), H - Y) / len(Y)

def cost(Theta, X, Y):
    H = np.dot(X, Theta)

    aux = (H - Y) ** 2

    return aux.sum() / (2 * len(X))

def gradient_reg(Theta, X, Y, Landa):
    G = gradient(Theta, X, Y)

    G[1:] += Landa / len(X) * Theta[1:]

    return G

def cost_reg(Theta, X, Y, Landa):
    C = cost(Theta, X, Y)

    aux2= np.sum(Theta[1:] ** 2)
    aux1 = (2 * len(X))

    aux = aux1 * aux2

    return C + Landa / aux

def cost_gradient_reg(Theta, X, Y, Landa):
    return [cost_reg(Theta, X, Y, Landa), gradient_reg(Theta, X, Y, Landa)]

def polynomial_features(X, p):
    for i in range(2, p + 1):
        aux = X[:, 0] ** i

        X = np.hstack((X, np.reshape(aux, (aux.shape[0], 1))))

    return X

def normalize(X):
    Mu = np.mean(X, axis=0)
    Sigma = np.std(X,axis=0)
    X_norm = (X - Mu) / Sigma

    return [X_norm, Mu, Sigma]

def test_hypotesis(norm_X, Y, X_test, Y_test, Mu, Sigma, Landa, p):
    new_X_test = polynomial_features(X_test, p)
    norm_X_test = (new_X_test - Mu) / Sigma
```

```
    norm_X_test = np.hstack((np.ones((len(norm_X_test), 1)), norm_X_test))

    Theta = np.ones(len(norm_X[0]))

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y, Landa),
method='TNC', jac=True)

    cost = cost_reg(fmin.x, norm_X_test, Y_test, Landa)

    print("With λ = {}, cost for X_text is: {}".format(Landa, cost))

def linear_regression_graph(X, Y, Theta, name):
    plt.figure()

    X_l = [np.amin(X), np.amax(X)]
    Y_l = [Theta[0] + Theta[1] * X_l[0], Theta[0] + Theta[1] * X_l[1]]

    plt.plot(X, Y, 'x', c='r')
    plt.plot(X_l, Y_l, c='b')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)

def learning_curve_graph(X1, X2, x_label, title, name):
    plt.figure()

    Y = range(len(X1))

    plt.title(title)
    plt.plot(Y, X1, c='orange', label="Train")
    plt.plot(Y, X2, c='blue', label="Cross Validation")
    plt.xlabel(x_label)
    plt.ylabel("Error")
    plt.legend()
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)

def polynomial_regression_graph(X, Y, Theta, Mu, Sigma, p, title, name):
    plt.figure()

    X_t = np.arange(np.amin(X) - 5, np.amax(X) + 5, 0.05)
    X_t = X_t.reshape(-1, 1)
    X_p = polynomial_features(X_t, p)
    X_p = (X_p - Mu) / Sigma
    X_p = np.hstack((np.ones((len(X_p), 1)), X_p))
    Y_t = np.dot(X_p, Theta)

    plt.title(title)
    plt.plot(X, Y, 'x', c='r')
    plt.plot(X_t, Y_t, c='b')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)

def learning_curve_graph2(X, Y1, Y2, x_label, title, name):
    plt.figure()
    plt.title(title)
    plt.plot(X, Y1, c='orange', label="Train")
    plt.plot(X, Y2, c='blue', label="Cross Validation")
    plt.xlabel(x_label)
    plt.ylabel("Error")
```

```
    plt.legend()
    plt.savefig('Práctica 5/Recursos/{}.png'.format(name), dpi=200)


# 1. Regresión Lineal Regularizada
def linear_regression(X, Y):
    X = np.hstack((np.ones((len(X), 1)), X))

    Theta = np.ones(2)
    Landa = 1

    cost, grad = cost_gradient_reg(Theta, X, Y, Landa)

    print("For Theta = {} and Lamda = {}:\n  Cost = {}\n  Gradient =
{}".format(Theta, Landa, cost, grad))

    Landa = 0

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(X, Y, Landa),
method='TNC', jac=True)

    linear_regression_graph(X[:,1:], Y, fmin.x, name="linear_regression")


# 2. Curva de Aprendizaje
def learning_curve(X, Y, X_val, Y_val, Landa, name="learning_curves"):
    m = len(X)

    X = np.hstack((np.ones((len(X), 1)), X))
    X_val = np.hstack((np.ones((len(X_val), 1)), X_val))

    Theta = np.ones(len(X[0]))

    cost = []
    cost_val = []

    for i in range(1, m + 1):
        fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(X[0:i],
Y[0:i], Landa), method='TNC', jac=True)

        cost.append(fmin['fun'])
        cost_val.append(cost_gradient_reg(fmin.x, X_val, Y_val, Landa)[0])

    learning_curve_graph(cost, cost_val, "Number of training examples",
"Learning curve for lineal regression (λ = {})".format(Landa), name=name)


# 3. Regresión Polinomial
def polinomial_regression(X, Y, X_val, Y_val, Landa, p):
    new_X = polynomial_features(X, p)
    norm_X, mu_X, sigma_X = normalize(new_X)
    norm_X = np.hstack((np.ones((len(norm_X), 1)), norm_X))

    Theta = np.ones(len(norm_X[0]))

    fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y, Landa),
method='TNC', jac=True)
```

```
    polynomial_regression_graph(X, Y, fmin.x, mu_X, sigma_X, p, "Learning
curve for lineal regression (λ = {})".format(Landa),
"polinomial_regression_graph")

    new_X_val = polynomial_features(X_val, p)
    norm_X_val = (new_X_val - mu_X) / sigma_X

    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=0,
name="learning_curves_2_λ0")
    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=1,
name="learning_curves_2_λ1")
    learning_curve(norm_X[:,1:], Y, norm_X_val, Y_val, Landa=100,
name="learning_curves_2_λ100")


# 4. Selección del Parámetro λ
def parameter_selection(X, Y, X_val, Y_val, X_test, Y_test, p):
    Landas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
    cost = []
    cost_val = []

    new_X = polynomial_features(X, p)
    norm_X, Mu, Sigma = normalize(new_X)
    norm_X = np.hstack((np.ones((len(norm_X), 1)), norm_X))

    new_X_val = polynomial_features(X_val, p)
    norm_X_val = (new_X_val - Mu) / Sigma
    norm_X_val = np.hstack((np.ones((len(norm_X_val), 1)), norm_X_val))

    Theta = np.ones(len(norm_X[0]))

    for i in Landas:
        fmin = minimize(fun=cost_gradient_reg, x0=Theta, args=(norm_X, Y, i),
method='TNC', jac=True)

        cost.append(fmin['fun'])
        cost_val.append(cost_reg(fmin.x, norm_X_val, Y_val, i))

    learning_curve_graph2(Landas, cost, cost_val, "Lambda", "Selecting λ
using a cross validation set", "selecting_lambda")

    test_hypotesis(norm_X, Y, X_test, Y_test, Mu, Sigma, 3, p)


def main():
    data = loadmat('Práctica 5/Recursos/ex5data1.mat')

    X, Y = data['X'], data['y'].ravel()
    X_val, Y_val = data['Xval'], data['yval'].ravel()
    X_test, Y_test = data['Xtest'], data['ytest'].ravel()
    Landa, p = 0, 8

    linear_regression(X, Y)
    learning_curve(X, Y, X_val, Y_val, Landa)
    polinomial_regression(X, Y, X_val, Y_val, Landa, p)
    parameter_selection(X, Y, X_val, Y_val, X_test, Y_test, p)


main()
```