Regresión lineal regularizada: sesgo y varianza

Código

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

data = loadmat('p4/ex5data1.mat')
y = data['y']
X = data['X']
yval = data['yval']
Xval = data['Xval']
ytest = data['ytest']
Xtest = data['Xtest']

size = len(y)
sizeval = len(yval)
X = X.reshape(X.shape[0])
Xval = Xval.reshape(Xval.shape[0])
X_new = np.ones((size, 2))
X_new[:,1] = X
Xval_new = np.ones((sizeval, 2))
Xval_new[:,1] = Xval

theta = np.zeros((2, 1))
iterations = 1500
alpha = 0.01

def cost_function(theta, X, y, lambda_reg):
    m = len(y)
    theta = theta.reshape((-1, 1))
    gradient = np.zeros(theta.shape)
    pred = X.dot(theta)
    sse = (pred - y) ** 2
    J = (1.0 / (2 * m)) * np.sum(sse) + (float(lambda_reg) / (2 * m)) *
np.sum(theta[1:] ** 2)
    gradient = (1.0 / m) * X.T.dot(pred - y) + (float(lambda_reg) / m) *
np.hstack((np.zeros((1, 1)), theta[1:].T))
    gradient = gradient[:, 0]
    return J, gradient

result = cost_function(np.ones((2, 1)), X_new, y, 1)
print("Cost = " + str(result[0]))
print("Gradient = " + str(result[1]))

thetaOptimized = opt.minimize(cost_function, theta, jac=True, args=(X_new, y, 0))
thetaOptimized = thetaOptimized.x

linspace = np.linspace(-40, 40, 1000)
plt.scatter(X, y)
plt.plot(linspace, thetaOptimized[0] + linspace*thetaOptimized[1], color='red')
plt.xlabel("Change in water level (x)")
plt.ylabel("Water flowing out of the dam (y)")
plt.show()
```

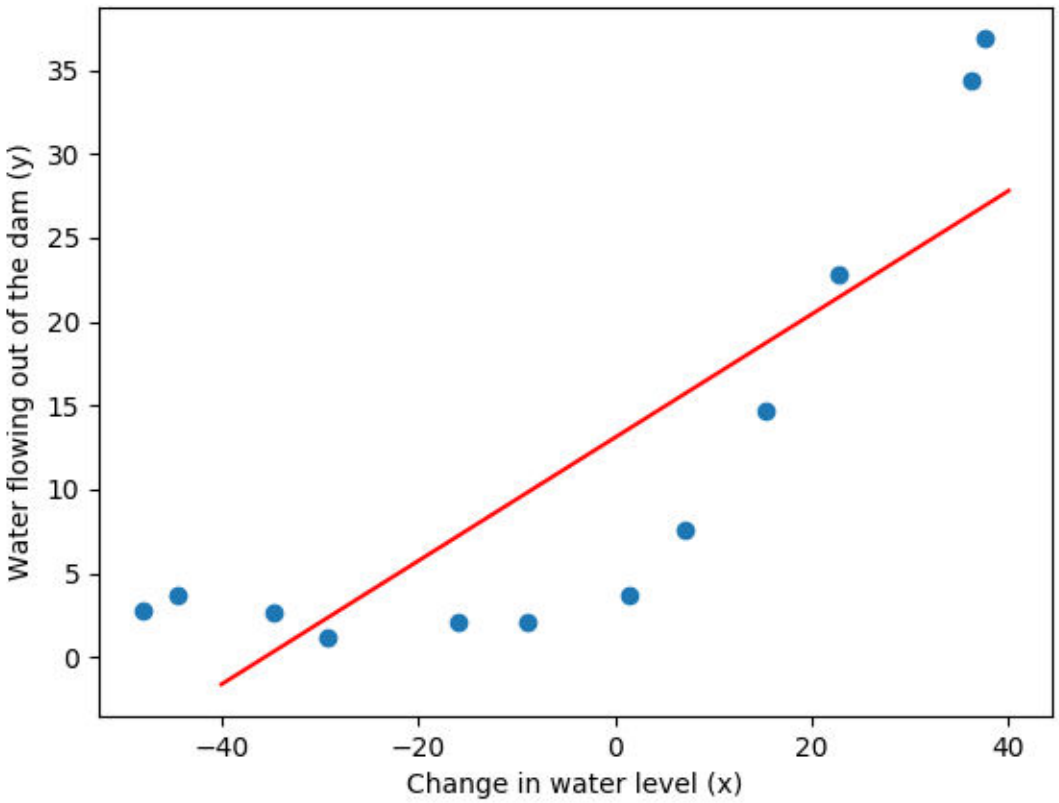
```
def cost_function_for_learning_curve(theta, X, y):
    m = len(y)
    theta = theta.reshape((-1, 1))
    pred = X.dot(theta)
    sse = (pred - y) ** 2
    J = (1.0 / (2 * m)) * np.sum(sse)
    return J

def learning_curve(X, y, Xval, yval, lambda_reg):
    m = len(y)
    trainError = np.zeros((m))
    valError = np.zeros((m))
    for i in range(1, m+1):
        theta = np.zeros((X.shape[1], 1))
        Xtrain = X[0:i,:]
        ytrain = y[0:i]
        thetaOptimized = opt.minimize(cost_function, theta, jac=True, method='TNC',
args=(Xtrain, ytrain, lambda_reg))
        thetaOptimized = thetaOptimized.x
        resultTrain = cost_function_for_learning_curve(thetaOptimized, Xtrain,
ytrain)
        resultVal = cost_function_for_learning_curve(thetaOptimized, Xval, yval)
        trainError[i-1] = resultTrain
        valError[i-1] = resultVal
    return trainError, valError

learningCurve = learning_curve(X_new, y, Xval_new, yval, 0)
plt.plot(range(1, size+1), learningCurve[0], color='red', label='Train')
plt.plot(range(1, size+1), learningCurve[1], color='blue', label='Cross
validation')
plt.xlabel("Number of training examples")
plt.ylabel("Error")
plt.title("Learning curve for linear regression")
plt.legend()
plt.show()

def polynomialFeatures(X, power):
      X_poly = np.zeros((len(X), power))
      Xcol = X.flatten()
      for i in range(1, power + 1):
            X_poly[:, i - 1] = Xcol ** i
      return X_poly

def normalize(X):
      mu = np.mean(X, axis=0)
      sigma = np.std(X, axis=0)
      normalized_X = (X - mu) / sigma
      return normalized_X, mu, sigma

def plotFit(min_x, max_x, mu, sigma, theta, p):
      x = np.arange(min_x - 5, max_x + 5, 0.05).reshape((-1, 1))
      x_poly = polynomialFeatures(x, p)
      x_poly = (x_poly - mu) / sigma
      x_poly = np.hstack((np.ones(len(x_poly)).reshape((-1, 1)), x_poly))
      plt.plot(x, np.matmul(x_poly, theta[:, None]), 'b-', linewidth=2)

p = 8
X_poly = polynomialFeatures(X, p)
X_poly, mu, sigma = normalize(X_poly)
```

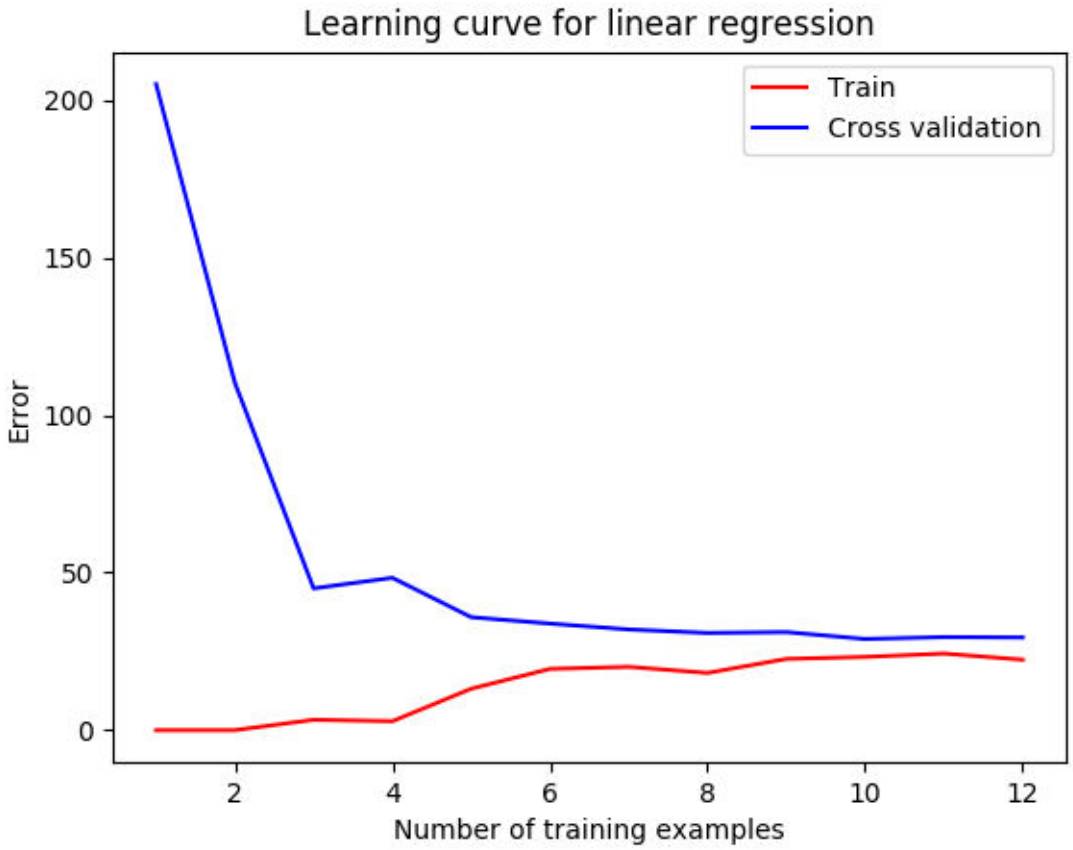
```
X_poly = np.hstack((np.ones_like(y), X_poly))

Xval_poly = polynomialFeatures(Xval, p)
for i in range(Xval_poly.shape[1]):
    Xval_poly[:, i] = (Xval_poly[:, i] - mu[i]) / sigma[i]
Xval_poly = np.hstack((np.ones_like(yval), Xval_poly))

Xtest_poly = polynomialFeatures(Xtest, p)
for i in range(Xtest_poly.shape[1]):
    Xtest_poly[:, i] = (Xtest_poly[:, i] - mu[i]) / sigma[i]
Xtest_poly = np.hstack((np.ones_like(ytest), Xtest_poly))

reg_param = 0.0
initial_theta = np.zeros(np.size(X_poly, 1))
res = opt.minimize(fun=cost_function,
            x0=initial_theta,
            args=(X_poly, y, reg_param),
            jac=True,
            method='TNC',
            options={'maxiter': 400, 'disp': False})
est_theta = res.x
plt.figure()
plt.plot(X, y, 'rx', markersize=10, linewidth=1.5)
plotFit(np.min(X), np.max(X), mu, sigma, est_theta, p)
plt.title("Polynomial regression (lambda = 0)")
plt.xlabel('Change in water level (x)')
plt.ylabel('Water flowing out of the dam (y)')
plt.show()

for lambda_reg in [0, 1, 100]:
    learningCurvePoly = learning_curve(X_poly, y, Xval_poly, yval, lambda_reg)
    plt.plot(range(1, size+1), learningCurvePoly[0], color='red', label='Train')
    plt.plot(range(1, size+1), learningCurvePoly[1], color='blue', label='Cross
validation')
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.title("Learning curve for linear regression (lambda = " + str(lambda_reg) +
")")
    plt.legend()
    plt.show()

lambdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
trainingErrors = []
validationErrors = []
for lambda_reg in lambdas:
    theta = np.zeros((X_poly.shape[1], 1))
    thetaOptimizedTrain = opt.minimize(cost_function, theta, jac=True,
args=(X_poly, y, lambda_reg), method='TNC', options={'maxiter': 400, 'disp':
False})
    thetaOptimizedTrain = thetaOptimizedTrain.x
    resultTrain = cost_function(thetaOptimizedTrain, X_poly, y, lambda_reg)[0]
    resultVal = cost_function(thetaOptimizedTrain, Xval_poly, yval, lambda_reg)[0]
    trainingErrors.append(resultTrain)
    validationErrors.append(resultVal)
plt.plot(lambdas, trainingErrors, color='red', label='Train')
plt.plot(lambdas, validationErrors, color='blue', label='Cross validation')
plt.xlabel("Lambda")
plt.ylabel("Error")
plt.ylim(0, 50)
```

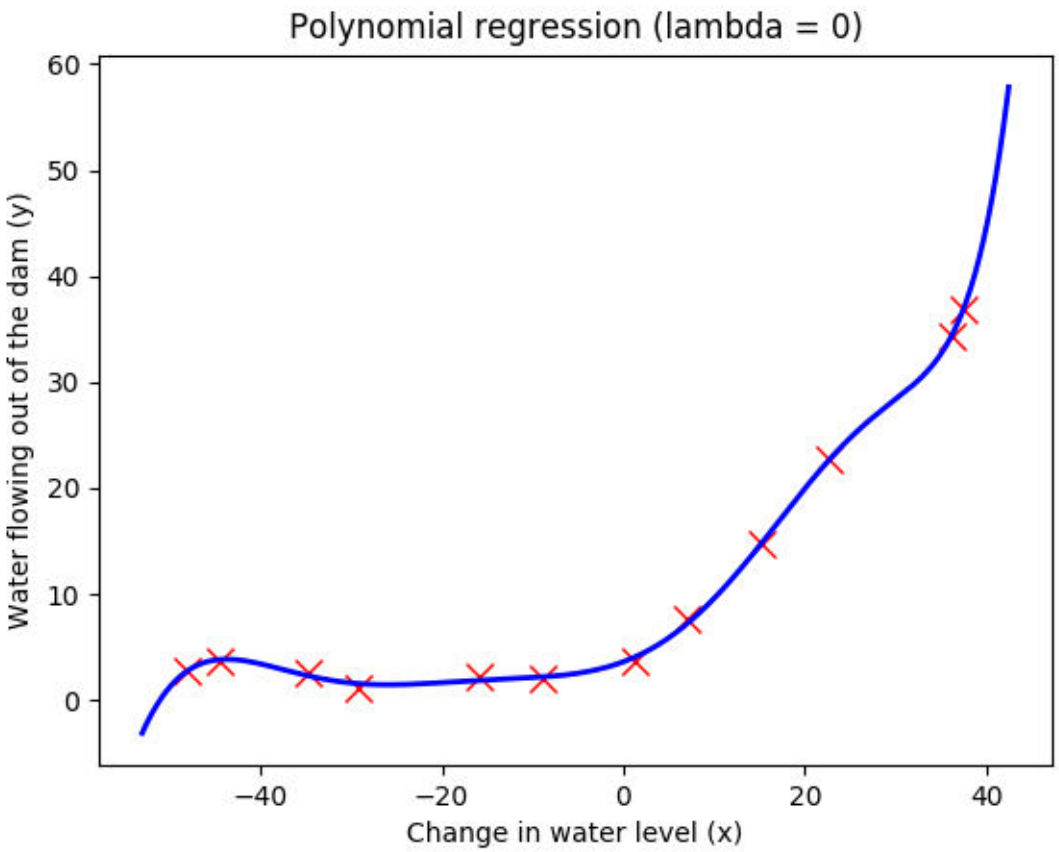
```
plt.title("Selecting lambda using a cross validation set")
plt.legend()
plt.show()

theta = np.zeros((X_poly.shape[1], 1))
thetaOptimizedTrain = opt.minimize(cost_function, theta, jac=True, args=(X_poly, y,
3), method='TNC', options={'maxiter': 400, 'disp': False})
thetaOptimizedTrain = thetaOptimizedTrain.x
resultTrain = cost_function(thetaOptimizedTrain, Xtest_poly, ytest, 3)[0]
print("Test error = " + str(resultTrain))
```

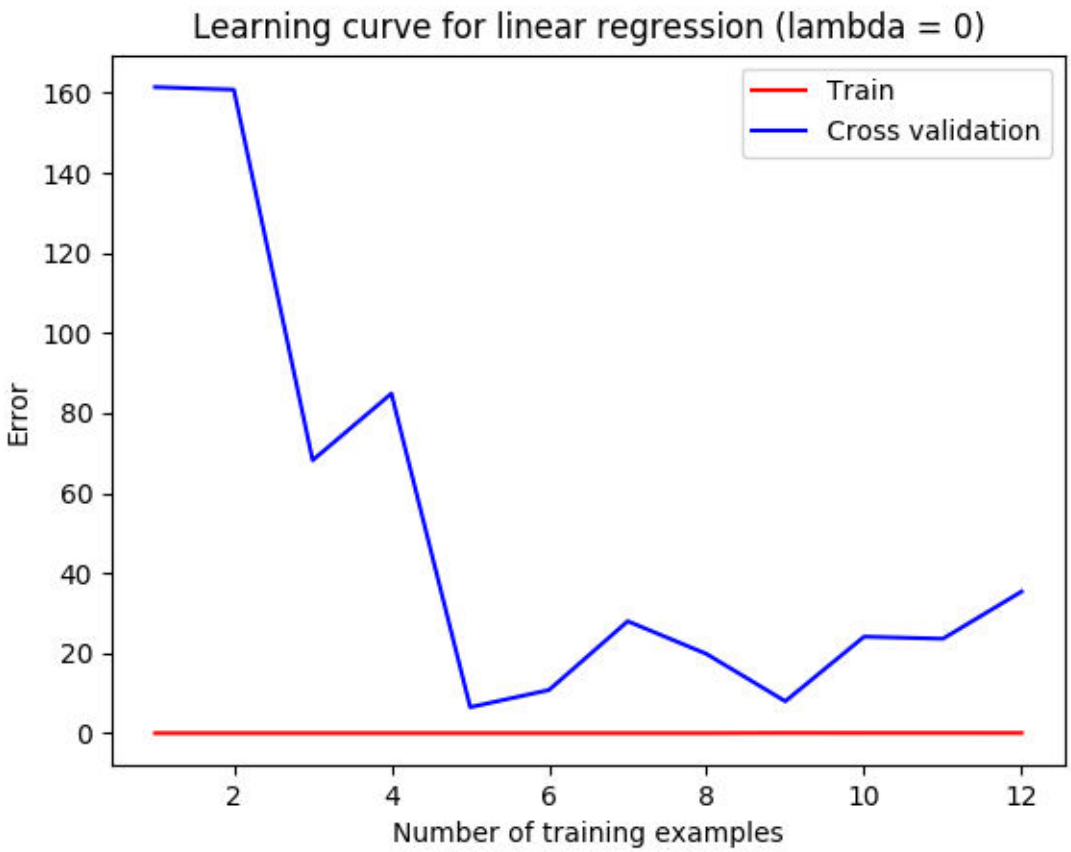
Resultados

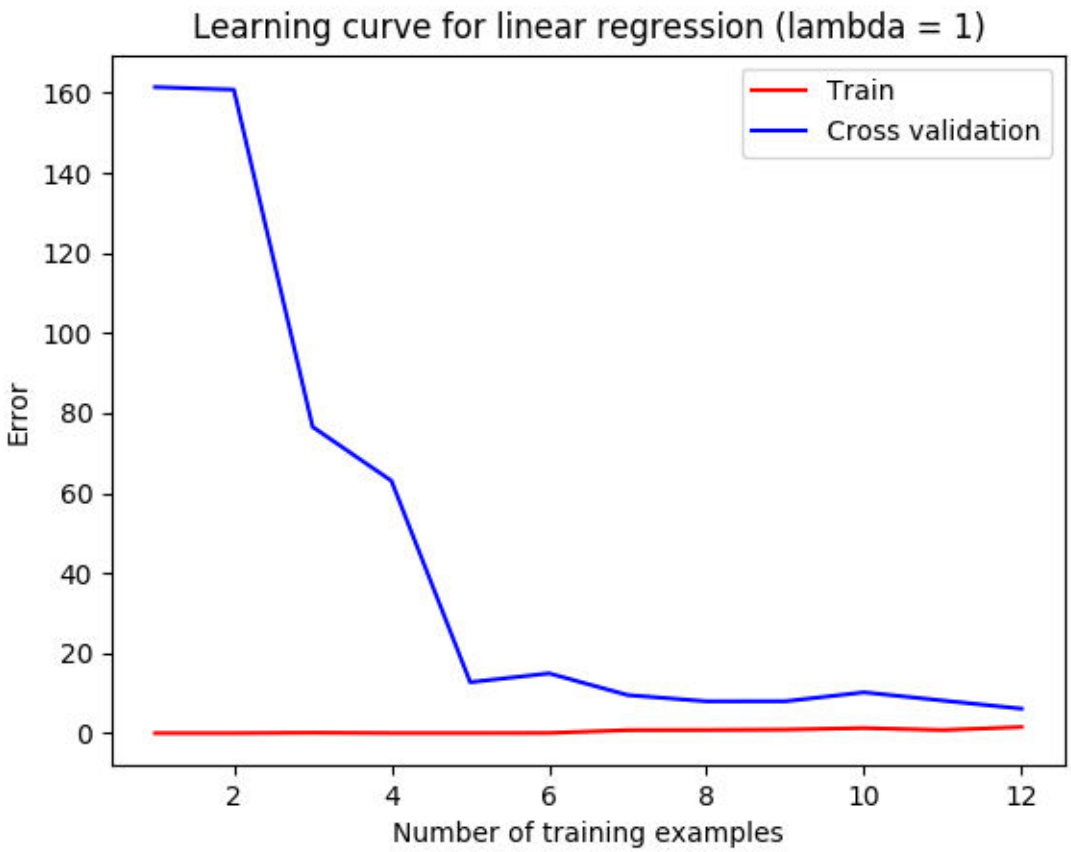
```
Cost = 303.9931922202643
Gradient = [-15.30301567 598.16741084]
```

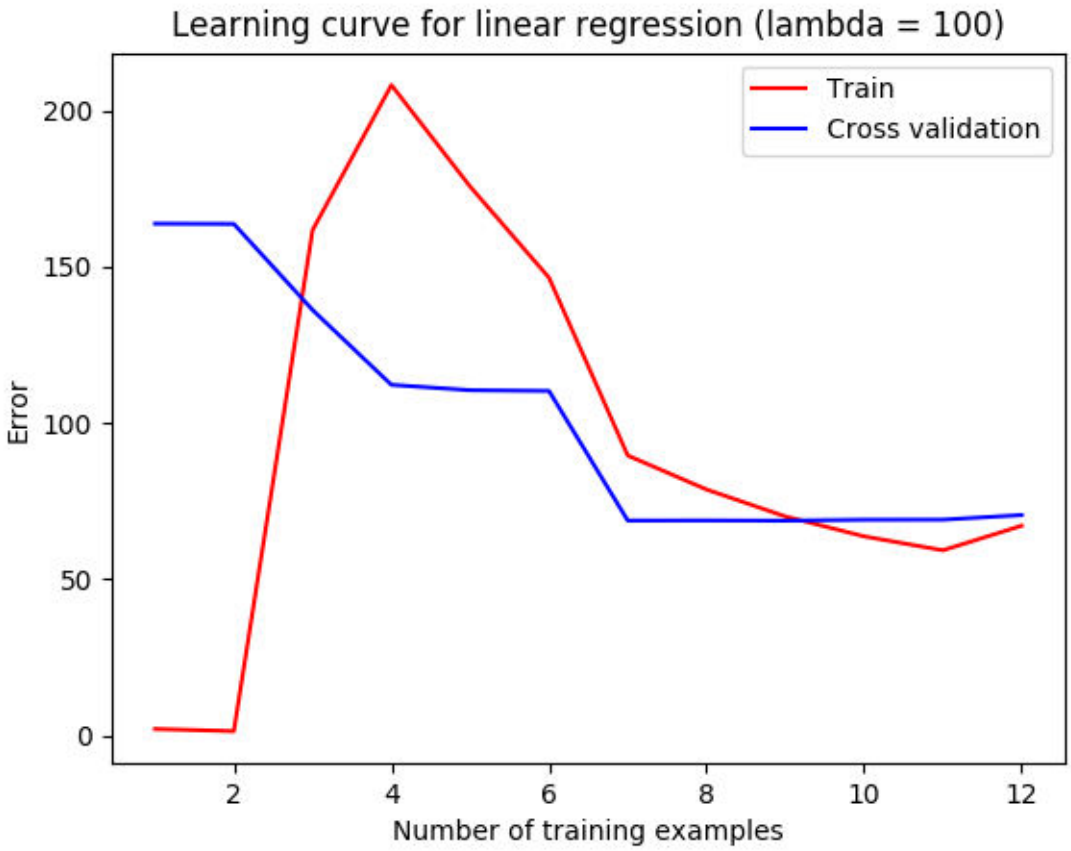


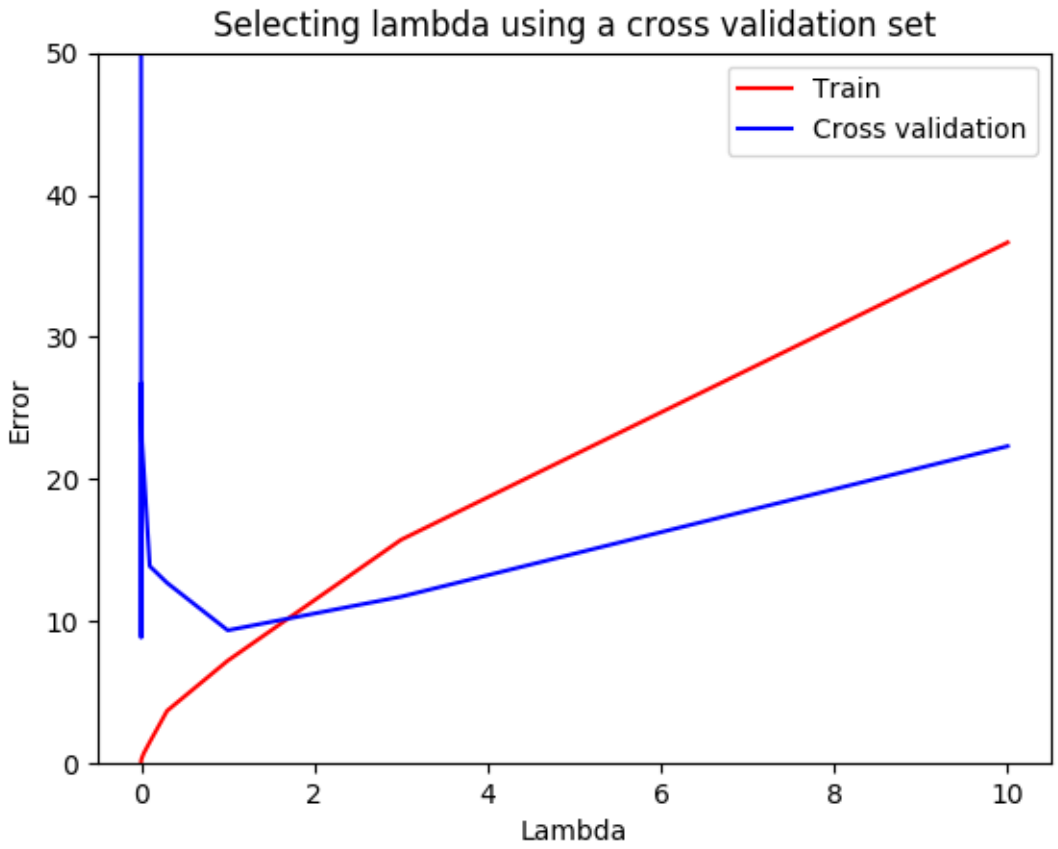

Learning curve for linear regression
Train
Cross validation
Error
200
150
100
50
0
2
4
6
8
10
12
Number of training examples


Polynomial regression (lambda = 0)
Water flowing out of the dam (y)
Change in water level (x)
60
50
40
30
20
10
0
−40
−20
0
20
40


Learning curve for linear regression (lambda = 0)
Train
Cross validation
Error
Number of training examples
0
20
40
60
80
100
120
140
160
2
4
6
8
10
12


Learning curve for linear regression (lambda = 1)
Train
Cross validation
Error
Number of training examples
0
20
40
60
80
100
120
140
160
2
4
6
8
10
12


Learning curve for linear regression (lambda = 100)
Train
Cross validation
Error
Number of training examples
0
50
100
150
200
2
4
6
8
10
12




```
Test error = 10.04616441432773
```

Comentarios

Para regresión lineal, cuando el número de ejemplos de entrenamiento se aumenta, la diferencia entre train error y test error se disminuye, ambos errores tienen valores pequeños. Para regresión polynomial el mejor valor de lambda está cerca de 1, para valores 0 y 100 el entrenamiento no funciona bien. La selección de lambda debería mostrar que el mejor valor de lambda es 3. Test error debería ser cerca de 3.