5 REGRESION LINEAL REGULARIZADA, SESGO Y VARIANZA

June 19, 2019

```
In [1]: #José Antonio Garido Sualdea
```

```
In [2]: ################################################
        #REGRESIÓN LINEAL REGULARIZADA, SESGO Y VARIANZA
        ################################################
```

```
In [3]: from scipy.io import loadmat
        import matplotlib.pyplot as plt
        from scipy.optimize import minimize
        import numpy as np
```

```
In [4]: def preprocess(X):
            return np.c_[np.ones(X.shape), X]
```

```
In [5]: #X, y de entrenamiento; Xval, yval de validación
        data = loadmat ('ex5data1.mat')
        X = preprocess(data['X'])
        y = np.ndarray.flatten(data['y'])
        Xval = preprocess(data['Xval'])
        yval = np.ndarray.flatten(data['yval'])
        Xtest = preprocess(data['Xtest'])
        ytest = np.ndarray.flatten(data['ytest'])
```

```
In [6]: #hipótesis, predicción del beneficio en función de la población
        def h(x, T): #t = teta
            return x.dot(T)

        #función de coste
        def J(T):
            (m, n) = X_.shape
            return ((X_.dot(T) - y_).transpose().dot(X_.dot(T) - y_)
                    / (2*m)) + (l/(2*m))*np.sum(T**2)

        #función de gradiente
        def gradiente(T):
            (m, n) = X_.shape
            T_ = T.copy()
```

```
        T_[0] = 0
        return (1/m)*X_.transpose().dot((X_.dot(T))-y_) + (1/m)*T_
```

```
In [7]: X_ = X
        y_ = y
        l = 1
        T = np.array([1,1])
        print('coste: {}'.format(J(T)))
        print('gradiente: {}'.format(gradiente(T)))
```

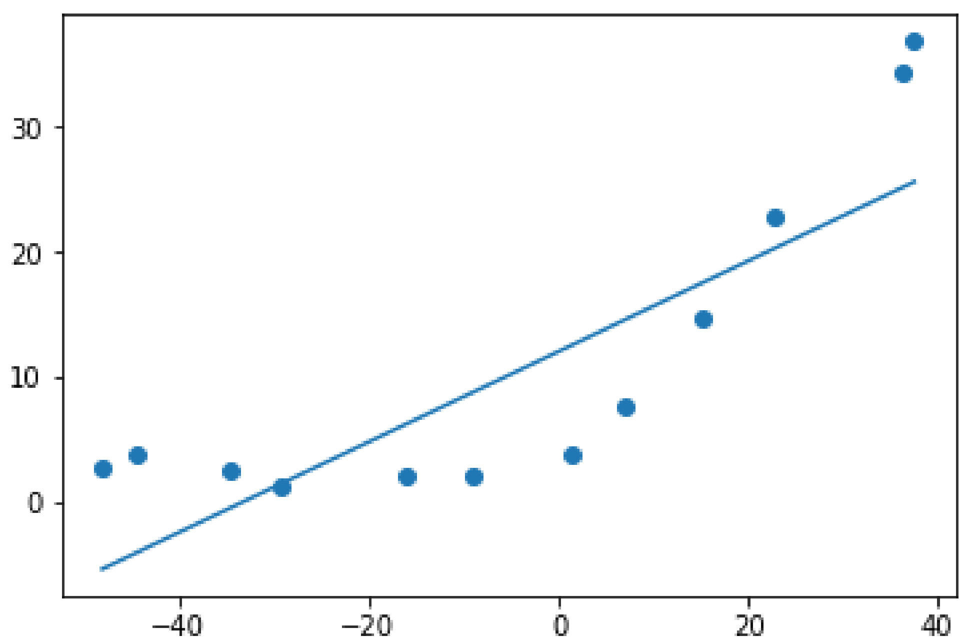
```
coste: 304.0348588869309
gradiente: [-15.30301567 598.25074417]
```

```
In [8]: def minimize_T():
            (m, n) = X_.shape
            T = np.ones(n)
            result = minimize(J, T)
            return result.x
```

```
In [9]: T_final = minimize_T()
```

```
In [10]: def plot_data(X, y, T): #función pintar gráfica
             fig = plt.figure()
             plt.scatter(x=X[:,1], y=y)
             plt.plot(sorted(X[:,1]),
                      [h(x, T) for x in sorted(X, key=lambda row: row[1])]
                     )
             plt.show()
```

```
In [11]: plot_data(X, y, T_final)
```



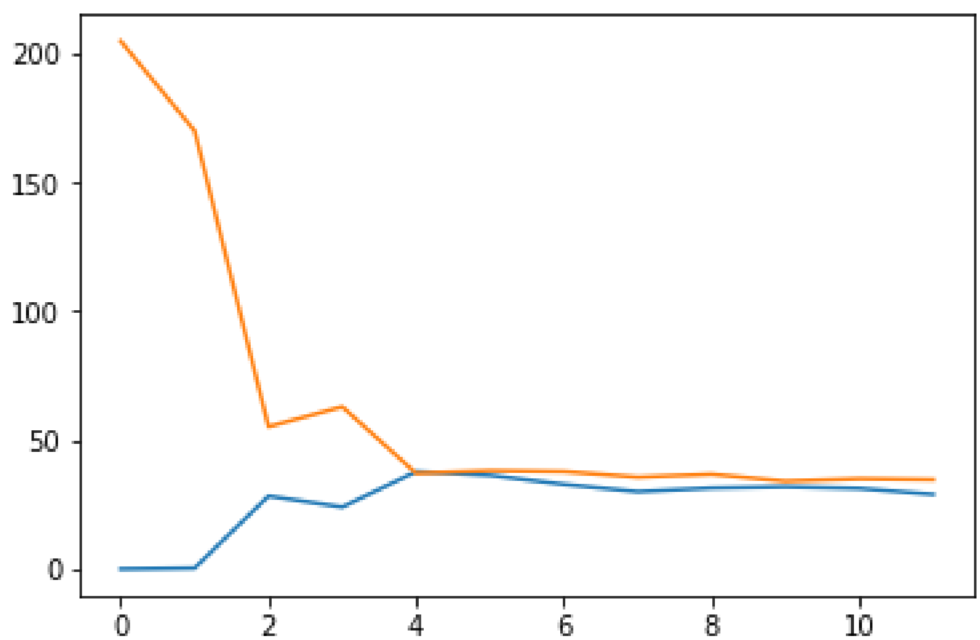
In [12]:
```
##########################   2
```

In [13]:
```
#función de coste para evaluación
def coste(X, y, T, l):
    (m, n) = X.shape
    return ((X.dot(T) - y).transpose().dot(X.dot(T) - y)
            / (2*m)) + (l/(2*m))*np.sum(T**2)

#función de error para evaluación
def error(X, y, T):
    (m, n) = X.shape
    return (np.absolute(X.dot(T) - y).sum() / m)
```

In [14]:
```
(m, n) = X.shape
coste_train = []
coste_val = []
for i in range(m):
    X_ = X[0:i + 1]
    y_ = y[0:i + 1]
    T_min = minimize_T()
    coste_train.append(coste(X_, y_, T_min, l))
    coste_val.append(coste(Xval, yval, T_min, l))
```

In [15]:
```
fig = plt.figure()
plt.plot(range(len(coste_train)), coste_train)
plt.plot(range(len(coste_val)), coste_val)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x1a0df7ef668>]
```



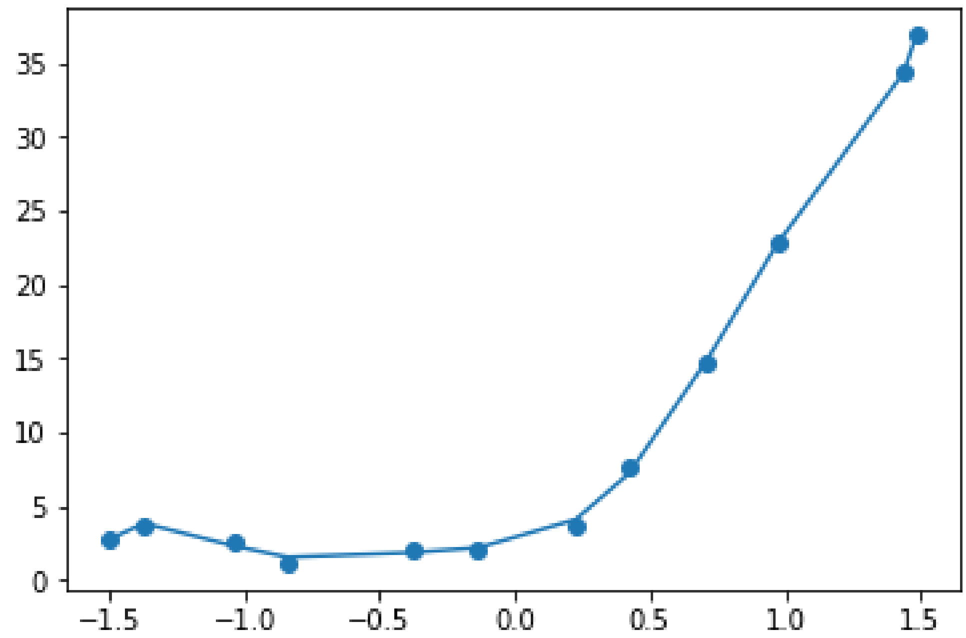
```
In [16]: ###################### 3
```

```
In [17]: def preprocess_pol(X, p):
             columns = [np.ones(X.shape), X]
             for i in range(2, p+1):
                 columns.append(X**i)
             return np.hstack(columns)
```

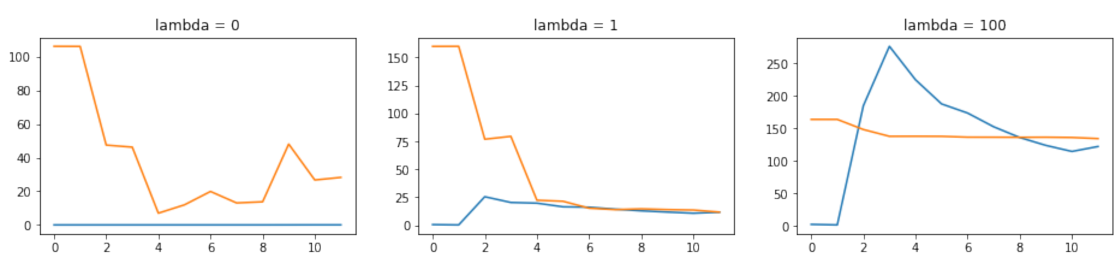
```
In [18]: def normalizar_columna(columna, mu, sigma):
             return (columna - mu) / sigma

         def normalizar(X):
             (m, n) = X.shape
             medias = []
             desviaciones = []
             for c in range(1, n):
                 column = X[:,c]
                 mu = column.mean()
                 sigma = column.std()
                 X[:,c] = normalizar_columna(column, mu, sigma)
                 medias.append(mu)
                 desviaciones.append(sigma)
             return X, medias, desviaciones
```

```
In [20]: X_p, medias, desviaciones = normalizar(preprocess_pol(data['X'], 8))
         Xval_p, medias_val, desviaciones_val =  normalizar(
                             preprocess_pol(data['Xval'], 8)
                             )
         Xtest_p, medias_test, desviaciones_test = normalizar(
                             preprocess_pol(data['Xtest'], 8)
                             )
```

```
In [21]: l = 0
         X_ = X_p
         T_final_p = minimize_T()
         plot_data(X_p, y, T_final_p)
```



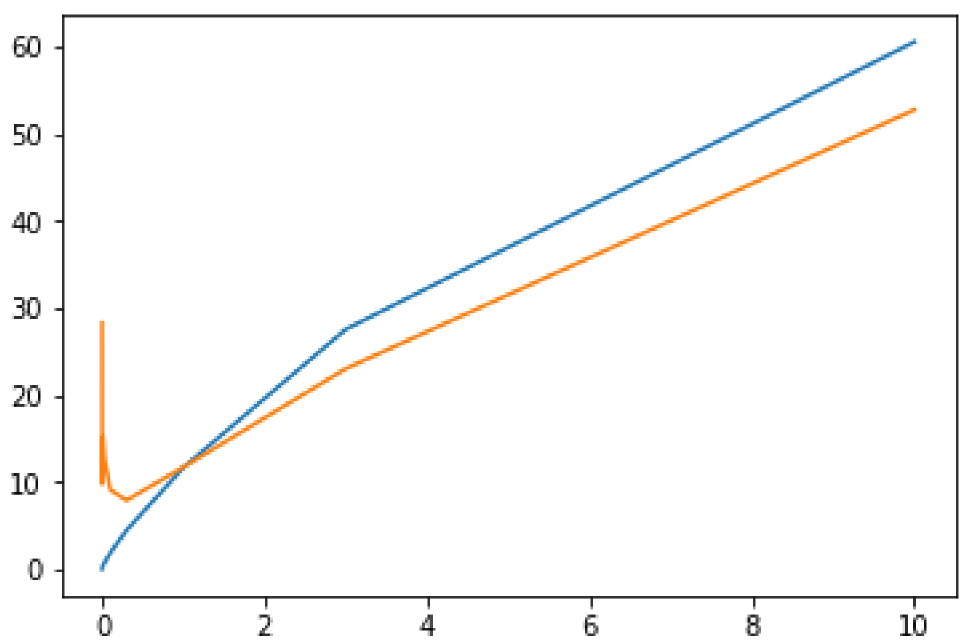

```
In [22]: fig, axs = plt.subplots(1,3,figsize=(16,3))
         lambdas = [0,1,100]
         (m, n) = X_p.shape
         for ax, l in zip(axs, lambdas):
             coste_train = []
             coste_val = []
             for i in range(m):
                 X_ = X_p[0:i + 1]
                 y_ = y[0:i + 1]
                 T_min = minimize_T()
                 coste_train.append(coste(X_, y_, T_min, l))
                 coste_val.append(coste(Xval_p, yval, T_min, l))
             ax.plot(range(len(coste_train)), coste_train)
             ax.plot(range(len(coste_val)), coste_val)
             ax.set_title('lambda = {}'.format(l))
```



In [23]: `#########################4`

In [24]:
```
fig = plt.figure()
lambdas = [0,0.001,0.003,0.01,0.03,0.1,0.3,1,3,10]
(m, n) = X_p.shape
coste_train = []
coste_val = []
X_ = X_p
y_ = y
for l in lambdas:
    T_min = minimize_T()
    coste_train.append(coste(X_, y_, T_min, l))
    coste_val.append(coste(Xval_p, yval, T_min, l))
plt.plot(lambdas, coste_train)
plt.plot(lambdas, coste_val)
```

Out[24]: [<matplotlib.lines.Line2D at 0x1a0dfbe59b0>]



In [25]:
```
fig = plt.figure()
l = 3
(m, n) = X_p.shape
X_ = X_p
y_ = y
T_min = minimize_T()
print(error(Xtest_p, ytest, T_min))
```

```
4.583693442059795
```

```
<Figure size 432x288 with 0 Axes>
```