# 2RegresionLogistica

June 19, 2019

In [70]:
```
#José Antonio Garrido Sualdea
```

In [71]:
```
####################
#REGRESIÓN LOGÍSTICA
####################
```

In [72]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.optimize as opt
```

In [73]:
```
#dataframe donde cargamos los datos
nombres_columnas = ['examen1', 'examen2', 'admision']
data = pd.read_csv('ex2data1.csv', names=nombres_columnas)
data.head()
```

Out[73]:

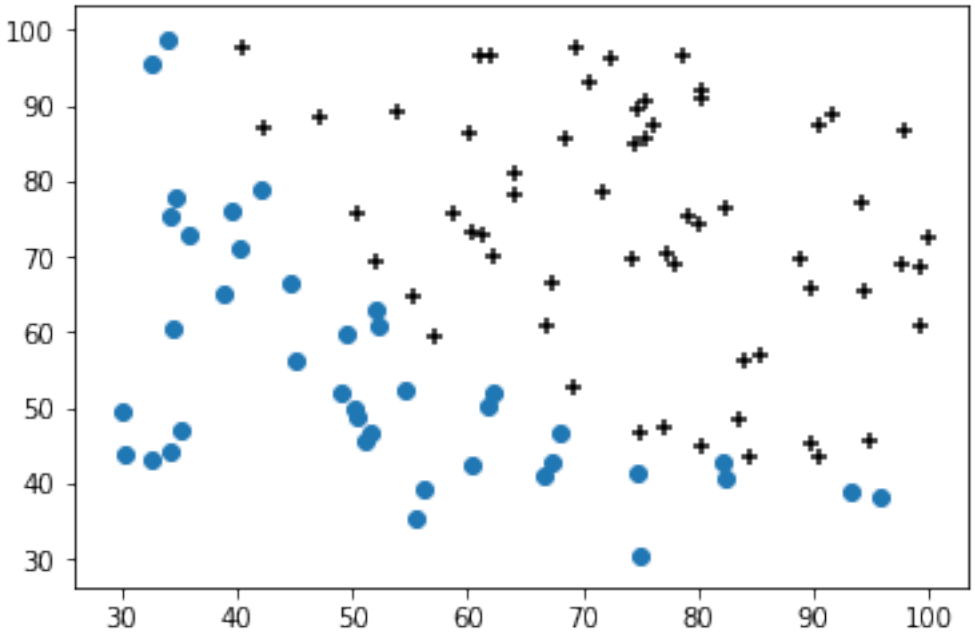
| | examen1 | examen2 | admision |
|---|---|---|---|
| 0 | 34.623660 | 78.024693 | 0 |
| 1 | 30.286711 | 43.894998 | 0 |
| 2 | 35.847409 | 72.902198 | 0 |
| 3 | 60.182599 | 86.308552 | 1 |
| 4 | 79.032736 | 75.344376 | 1 |

In [74]:
```
X = data[['examen1', 'examen2', ]].values
Y = data['admision'].values
(m, n) = X.shape
X = np.c_[X, np.ones(m)]
```

In [75]:
```
# Obtiene un vector con los índices de los ejemplos positivos
posPos = np.where (Y == 1)
# Dibuja los ejemplos positivos
plt.scatter(X[posPos, 0], X[posPos, 1], marker='+', c='k')

# Obtiene un vector con los índices de los ejemplos negativos
posNeg = np.where (Y == 0)
# Dibuja los ejemplos negativos
plt.scatter(X[posNeg, 0], X[posNeg, 1], marker='o')
```

Out[75]:
```
<matplotlib.collections.PathCollection at 0x1f844737b00>
```




In [76]:
```
#función sigmoide
def g(z):
    return 1/(1 + np.exp(-z))

#función de coste
def coste(T, X, Y, m):
    return -(1/m)*((np.log(g(X.dot(T))).transpose().dot(Y)
           + (np.log(1-g(X.dot(T)))).transpose().dot(1-Y)))

#función de gradiente
def gradiente(T, X, Y, m):
    return (1/m)*X.transpose().dot((g(X.dot(T))-Y))
```

In [77]:
```
T = np.array([0,0,0])
print(coste(T, X, Y, m))
print(gradiente(T, X, Y, m))
```

```
0.6931471805599452
[-12.00921659 -11.26284221  -0.1       ]
```

In [78]:
```
#para obtener el valor de los parámetros THETA
#que minimizan la función de coste
result = opt.fmin_tnc(
                        func=coste,
                        x0=np.array([0,0,0]),
```

```
                                fprime=gradiente, args=(X, Y, m)
                               )
        theta_opt = result[0]
```

```
In [79]: print(theta_opt)
         print(coste(theta_opt, X, Y, m)) #coste óptimo
```

```
[  0.20623159   0.20147149 -25.16131856]
0.20349770158947494
```

```
In [80]: n_aciertos = 0
         for i in range(m):
             prediccion = 0
             if g(X[i].dot(theta_opt)) > 0.5:
                 prediccion = 1
             if prediccion == Y[i]:
                 n_aciertos += 1

         print('{} %'.format(100 * n_aciertos / m))
```

```
89.0 %
```

```
In [81]: ###################################
         #REGRESIÓN LOGÍSTICA REGULARIZADA
         ###################################
```

```
In [82]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import scipy.optimize as opt
         from sklearn.preprocessing import PolynomialFeatures
```

```
In [83]: #dataframe donde cargamos los datos
         nombres_columnas = ['test1', 'test2', 'pasa']
         data = pd.read_csv('ex2data2.csv', names=nombres_columnas)
         data.head()
```

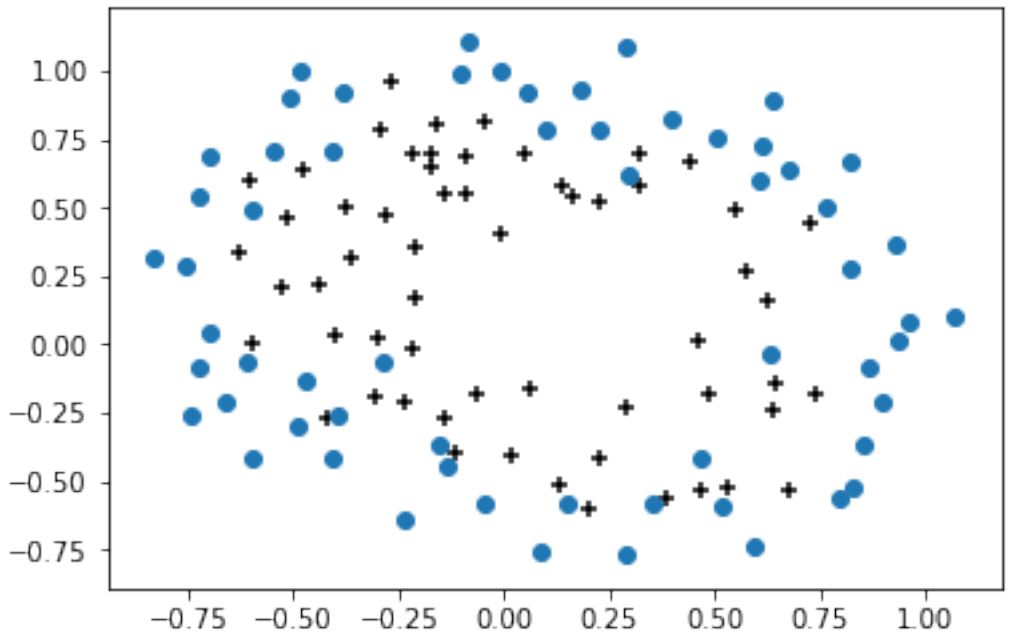
```
Out[83]:       test1    test2  pasa
         0  0.051267  0.69956     1
         1 -0.092742  0.68494     1
         2 -0.213710  0.69225     1
         3 -0.375000  0.50219     1
         4 -0.513250  0.46564     1
```

```
In [84]: X = data[['test1', 'test2', ]].values
         Y = data['pasa'].values
```

In [85]:
```
# Obtiene un vector con los índices de los ejemplos positivos
posPos = np.where (Y == 1)
# Dibuja los ejemplos positivos
plt.scatter(X[posPos, 0], X[posPos, 1], marker='+', c='k')

# Obtiene un vector con los índices de los ejemplos negativos
posNeg = np.where (Y == 0)
# Dibuja los ejemplos negativos
plt.scatter(X[posNeg, 0], X[posNeg, 1], marker='o')
```

Out[85]: <matplotlib.collections.PathCollection at 0x1f8447a4710>




In [86]:
```
pf = PolynomialFeatures(degree=6)
X = pf.fit_transform(X)
(m, n) = X.shape
```

In [87]:
```
#función sigmoide
def g(z):
    return 1/(1 + np.exp(-z))

#función de coste
def coste(T, X, Y, m, l):
    return -(1/m)*(
            (np.log(g(X.dot(T))).transpose().dot(Y)
             + (np.log(1-g(X.dot(T)))).transpose().dot(1-Y))
            ) + (l/(2*m))*np.sum(T**2)
```

```
        #función de gradiente
        def gradiente(T, X, Y, m, l):
            T_ = T.copy()
            T_[0] = 0
            return (1/m)*X.transpose().dot((g(X.dot(T))-Y)) + (l/m)*T_
```

```
In [88]: T = np.array([0]*n)
         l = 1
         print(coste(T, X, Y, m, l))
         print(gradiente(T, X, Y, m, l))
```

```
0.6931471805599453
[8.47457627e-03 1.87880932e-02 7.77711864e-05 5.03446395e-02
 1.15013308e-02 3.76648474e-02 1.83559872e-02 7.32393391e-03
 8.19244468e-03 2.34764889e-02 3.93486234e-02 2.23923907e-03
 1.28600503e-02 3.09593720e-03 3.93028171e-02 1.99707467e-02
 4.32983232e-03 3.38643902e-03 5.83822078e-03 4.47629067e-03
 3.10079849e-02 3.10312442e-02 1.09740238e-03 6.31570797e-03
 4.08503006e-04 7.26504316e-03 1.37646175e-03 3.87936363e-02]
```

```
In [89]: result = opt.fmin_tnc(
                              func=coste,
                              x0=np.array([0]*n),
                              fprime=gradiente, args=(X, Y, m, l)
                             )
         theta_opt = result[0]
```

```
In [90]: print(theta_opt)
         print(coste(theta_opt, X, Y, m, l)) #coste óptimo
```

```
[ 1.25441469  0.62276766  1.19242768 -2.00505525 -0.87290718 -1.36184269
  0.12573857 -0.35536843 -0.35603623 -0.17096338 -1.45784634 -0.06683329
 -0.61498633 -0.25080271 -1.18096021 -0.2256683  -0.20562885 -0.0638233
 -0.27187029 -0.27658993 -0.46836026 -1.03247356  0.01627586 -0.29695277
  0.00581073 -0.32631633 -0.12073381 -0.9302299 ]
0.5357749521779849
```

```
In [91]: n_aciertos = 0
         for i in range(m):
             prediccion = 0
             if g(X[i].dot(theta_opt)) > 0.5:
                 prediccion = 1
             if prediccion == Y[i]:
                 n_aciertos += 1

         print('{} %'.format(100 * n_aciertos / m))
```

```
83.05084745762711 %
```