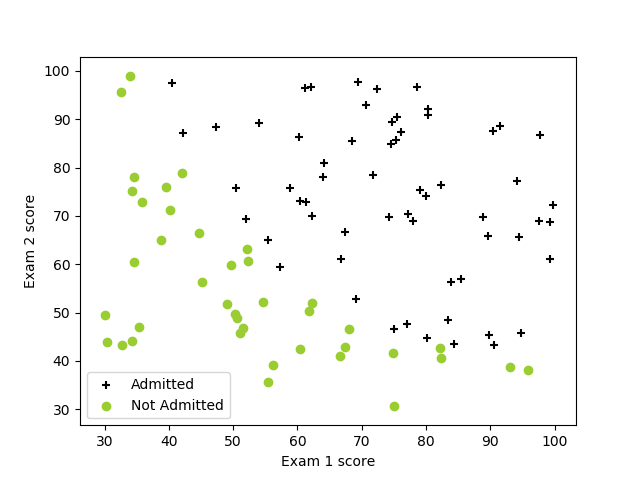
Memoria Técnica. Práctica 2

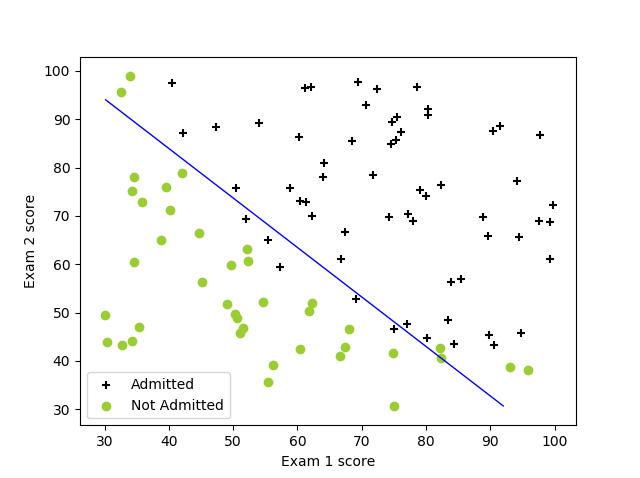
Sergio Gavilán Fernández [sgavil01@ucm.es](mailto:sgavil01@ucm.es)

Alejandro Villar Rubio [alvill04@ucm.es](mailto:alvill04@ucm.es)

PARTE 1: Regresión logística

Partiendo del siguiente conjunto de datos:



Hemos utilizado la función sigmoide para poder implementar la función de coste y su gradiente, de esta forma hemos hecho uso de *scipy.optimize.fmin\_tnc* para calcular la [](https://www.codecogs.com/eqnedit.php?latex=%5Ctheta%1) que minimiza la función de coste. De este modo podemos dibujar la frontera de la regresión logística: 

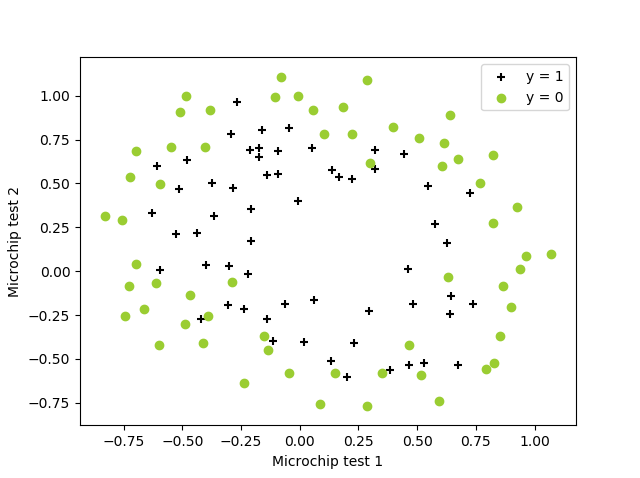
Posteriormente hemos calculado el porcentaje de ejemplos de entrenamiento que se clasifican correctamente, obteniendo un 89% de acierto.

El código de la parte 1 es el siguiente:

|  |
| --- |
| import numpy as np from pandas.io.parsers import read\_csv import matplotlib.pyplot as plt import scipy.optimize as opt  # Carga el fichero csv especificado y lo devuelve en un array de numpy  def carga\_csv(file\_name):  valores = read\_csv(file\_name, header=None).values  # suponemos que siempre trabajaremos con float  return valores.astype(float)   def dibuja\_grafica(X, Y):  admitted = np.where(Y == 1)  notAdmitted = np.where(Y == 0)   plt.scatter(X[admitted, 0], X[admitted, 1],  marker='+', c='k', label='Admitted')  plt.scatter(X[notAdmitted, 0], X[notAdmitted, 1], marker='o',  c='yellowgreen', label='Not Admitted')   plt.legend()   plt.xlabel('Exam 1 score')  plt.ylabel('Exam 2 score')   return plt   def dibuja\_h(Theta, X, Y, plt):  x1\_min, x1\_max = X[:, 0].min(), X[:, 0].max()  x2\_min, x2\_max = X[:, 1].min(), X[:, 1].max()   xx1, xx2 = np.meshgrid(np.linspace(x1\_min, x1\_max),  np.linspace(x2\_min, x2\_max))   h = sigmoide(np.c\_[np.ones((xx1.ravel().shape[0], 1)),  xx1.ravel(), xx2.ravel()].dot(Theta))  h = h.reshape(xx1.shape)   # el cuarto parámetro es el valor de z cuya frontera se  # quiere pintar  plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')  plt.show()  plt.savefig('parte1.jpg')  plt.close()   def funcion\_coste(Theta, X, Y):  m = len(X)  H = sigmoide(np.matmul(X, Theta))  return ((-1/m) \* (np.dot(np.log(sigmoide(np.dot(X, Theta))).T, Y) +  (np.dot(np.log(1 - sigmoide(np.dot(X, Theta))).T, 1 - Y))))   def funcion\_gradiente(Theta, X, Y):  m = X.shape[0]  return (1 / m) \* (np.dot(X.T, sigmoide(np.dot(X, Theta)) - Y))   def h(X, Theta):  return (1 / (1 + np.exp(np.dot(X, -Theta.T))))   def sigmoide(z):  return (1 / (1 + np.exp(-z))) # z = theta.T \* x   def regresion\_logistica(Theta, X, Y):  gradiente = funcion\_gradiente(Theta, X, Y)  coste = funcion\_coste(Theta, X, Y)   print("Función gradiente:", gradiente)  print("Función coste:", coste)   result = opt.fmin\_tnc(func=funcion\_coste, x0=Theta,  fprime=funcion\_gradiente, args=(X, Y))  Theta\_Opt = result[0]   return Theta\_Opt   def ev\_regresion(X, Y, Theta):  X\_ev = sigmoide(np.dot(X, Theta))  X\_ev = (X\_ev < 0.5)  Y\_ev = (Y == 0)  res = (X\_ev == Y\_ev)   porcentaje = ((np.sum(res)) \* 100) / res.shape  return porcentaje   def main():  datos = carga\_csv("ex2data1.csv")  X = np.delete(datos, np.shape(datos)[1]-1, axis=1)  Y = datos[:, datos.shape[1]-1]   plt = dibuja\_grafica(X, Y)   onesColumn = np.ones((X.shape[0], 1))  X = np.hstack((onesColumn, X))   Theta = np.zeros(X.shape[1])   Theta = regresion\_logistica(Theta, X, Y)  dibuja\_h(Theta, np.delete(X, 0, axis=1), Y, plt)   correctos = ev\_regresion(X, Y, Theta)  print("El", correctos[0], "% se clasifican correctamente.")   main() |

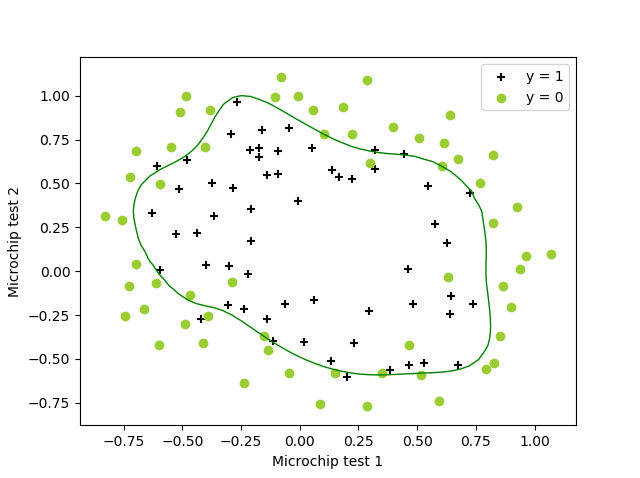
PARTE 2: Regresión logística regularizada

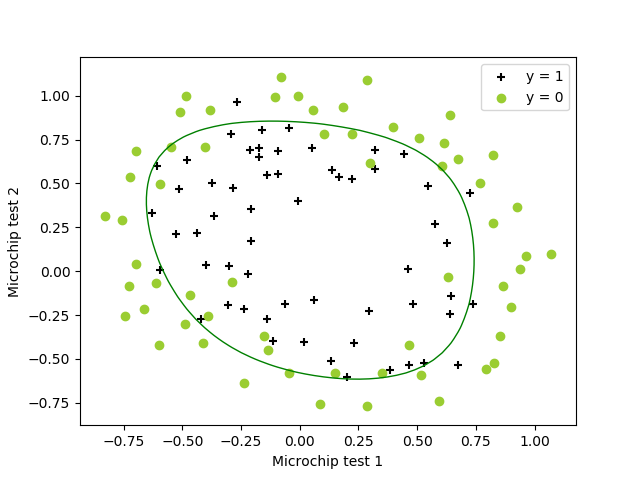
Partiendo de estos datos sobre microchips que superan o no un control de calidad:



En primer lugar hemos mapeado los atributos combinando los atributos originales para obtener unos mejores resultados, después, hemos calculado la función de coste y su gradiente para posteriormente calcular la [](https://www.codecogs.com/eqnedit.php?latex=%5Ctheta%1)óptima utilizando *scipy.optimize.fmin\_tnc.*

Finalmente se ha experimentado con distintos valores de lambda ( lambda = 0,lambda = 1)





Podemos observar que a mayor lambda la frontera se ajusta menos a los ejemplos de entrenamiento pero generará predicciones más ajustadas ( dentro de un rango )

El código de la parte 2 es el siguiente:

|  |
| --- |
| import numpy as np from pandas.io.parsers import read\_csv import matplotlib.pyplot as plt import scipy.optimize as opt from sklearn.preprocessing import PolynomialFeatures  # Carga el fichero csv especificado y lo devuelve en un array de numpy   def carga\_csv(file\_name):  valores = read\_csv(file\_name, header=None).values  # suponemos que siempre trabajaremos con float  return valores.astype(float)   def dibuja\_grafica(X, Y):  admitted = np.where(Y == 1)  notAdmitted = np.where(Y == 0)   plt.scatter(X[admitted, 0], X[admitted, 1],  marker='+', c='k', label='y = 1')  plt.scatter(X[notAdmitted, 0], X[notAdmitted, 1], marker='o',  c='yellowgreen', label='y = 0')   plt.legend()   plt.xlabel('Microchip test 1')  plt.ylabel('Microchip test 2')   return plt   def dibuja\_h(Theta, X, Y, plt, poly):  x1\_min, x1\_max = X[:, 0].min(), X[:, 0].max()  x2\_min, x2\_max = X[:, 1].min(), X[:, 1].max()   xx1, xx2 = np.meshgrid(np.linspace(x1\_min, x1\_max),  np.linspace(x2\_min, x2\_max))   h = sigmoide(poly.fit\_transform(  np.c\_[xx1.ravel(), xx2.ravel()]).dot(Theta))  h = h.reshape(xx1.shape)   # el cuarto parámetro es el valor de z cuya frontera se  # quiere pintar  plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='g')  plt.show()  plt.close()   def f\_gradiente(Theta, X, Y, lam):  m = len(X)  tempTheta = np.r\_[[0], Theta[1:]]  return (((1 / m) \* np.dot(X.T, sigmoide(np.dot(X, Theta)) - Y))  + ((lam / m) \* tempTheta))   def f\_coste(Theta, X, Y, lam):  m = len(X)  return (((-1 / m) \* (np.dot(np.log(sigmoide(np.dot(X, Theta))).T, Y)  + np.dot(np.log(1 - sigmoide(np.dot(X, Theta))).T, (1 - Y))))  + ((lam / (2 \* m)) \* np.sum(Theta\*\*2, initial=1)))   def sigmoide(z):  return 1 / (1 + np.exp(-z))   def regresion\_logistica\_regularizada(X, Y, Theta, lam):  poly = PolynomialFeatures(6)  X\_poly = poly.fit\_transform(X)   grad = f\_gradiente(Theta, X\_poly, Y, lam)  coste = f\_coste(Theta, X\_poly, Y, lam)   result = opt.fmin\_tnc(func=f\_coste, x0=Theta,  fprime=f\_gradiente, args=(X\_poly, Y, lam))  Theta\_Opt = result[0]  return poly, Theta\_Opt   def main():  datos = carga\_csv("ex2data2.csv")  X = np.delete(datos, np.shape(datos)[1]-1, axis=1)  Y = datos[:, datos.shape[1]-1]   Theta = np.zeros(28)  lam = 1   plt = dibuja\_grafica(X, Y)  poly, Theta\_Opt = regresion\_logistica\_regularizada(X, Y, Theta, lam)   dibuja\_h(Theta\_Opt, X, Y, plt, poly)   main() |