Práctica 6 – Support Vector Machines

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

  En esta práctica, crearemos un clasificador para detectar si un correo podría ser clasificado como 'spam' o no. Para ello, utilizaremos el clasificador SVM de la biblioteca sckit -learn. En la primera parte de la práctica jugaremos con el clasificador y con unos datos de entrenamiento para entender su funcionamiento, mientras que en la segunda parte haremos el clasificador de spam.

- Código de la práctica- Parte 1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt
import scipy.io as io
import sklearn.preprocessing
from sklearn.svm import SVC
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
import process_email, get_vocab_dict
from process_email import *
from get_vocab_dict import *
import codecs
```

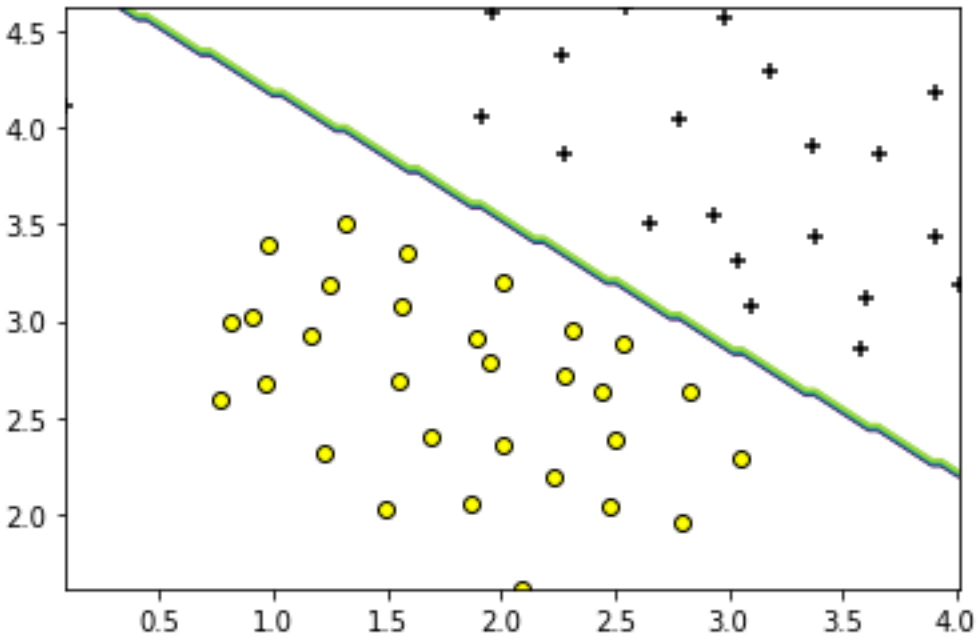
```
def visualize_boundary(X, y, svm, file_name=''):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1
.shape)
    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(
    X[neg, 0], X[neg, 1], color='yellow', edgecolors='black', marker=
'o')
    plt.contour(x1, x2, yp)
    plt.show()
    plt.close()
```

```
def selectCandSigma(X,Y,Xval,Yval):
    parameters = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    bestC = 0
    bestSigma = 0
    bestScore = 0
    for C in parameters:
        for sigma in parameters:
            svm = SVC(kernel='rbf', C=C, gamma=1 / ( 2 * sigma **2))
            svm.fit(X, Y.ravel())
            score = accuracy_score(Yval, svm.predict(Xval))
            if(bestScore < score):
                bestSigma = sigma
                bestScore = score
                bestC = C

    return bestC, bestSigma, bestScore
data1, data2, data3 = io.loadmat('./p6/ex6data1.mat'), io.loadmat('./p
6/ex6data2.mat'), io.loadmat('./p6/ex6data3.mat')
    X1, Y1 = data1['X'], data1['y']
    X2, Y2 = data2['X'], data2['y']
    X3, Y3 = data3['X'], data3['y']
    X3val, Y3val = data3['Xval'], data3['yval']

    #1.1. Kernel lineal
    svm = SVC(kernel='linear', C=1.0)
    svm.fit(X1, Y1.ravel())
    visualize_boundary(X1,Y1,svm)

    svm = SVC(kernel='linear', C=100.0)
    svm.fit(X1, Y1.ravel())
    visualize_boundary(X1,Y1,svm)

    #1.2. Kernel gaussiano
    sigma = 0.1
    svm = SVC(kernel='rbf', C=1, gamma=1 / ( 2 * sigma **2))
    svm.fit(X2, Y2.ravel())
    visualize_boundary(X2,Y2,svm)

    #1.3. Elección de los parámetros C y sigma
    C, sigma, score = selectCandSigma(X3,Y3,X3val,Y3val)
    svm = SVC(kernel='rbf', C=C, gamma=1 / ( 2 * sigma **2))
    svm.fit(X3, Y3.ravel())
    print('C=' + str(C) + ' BestSigma =' + str(sigma))
    visualize_boundary(X3,Y3,svm)
```

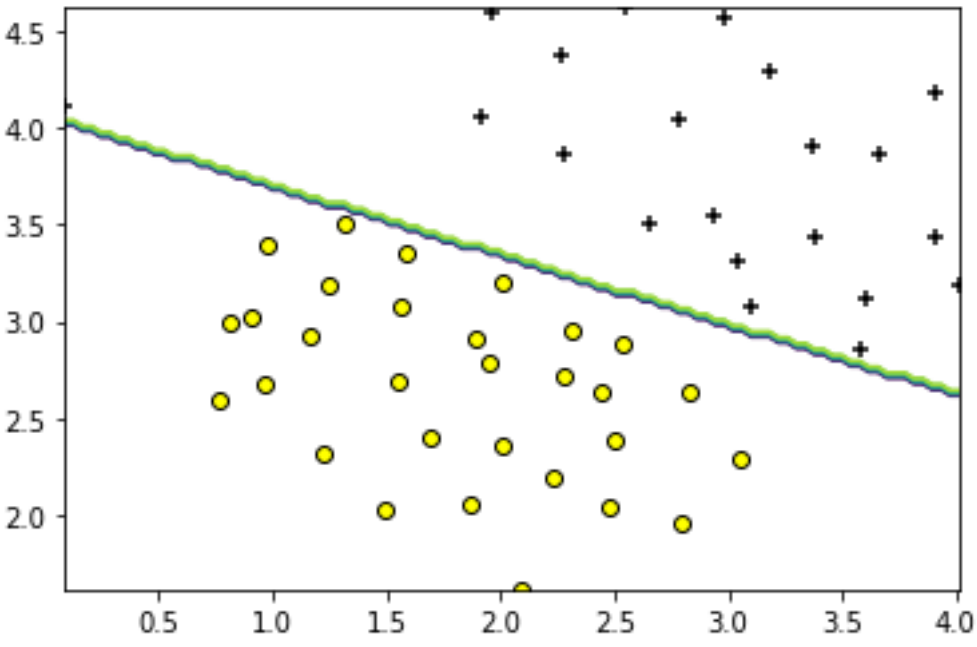
- Resultados de ejecución: parte 1

Tras leer los datos, se prueba a usar el clasificador SVM usando distintos valores del parámetro de regularización c. En primer lugar, se prueba un conjunto de datos linealmente separados, por lo que se usa un kernel lineal.

En esta primera parte, comprobamos el efecto de probar con C=1 y con C=100.

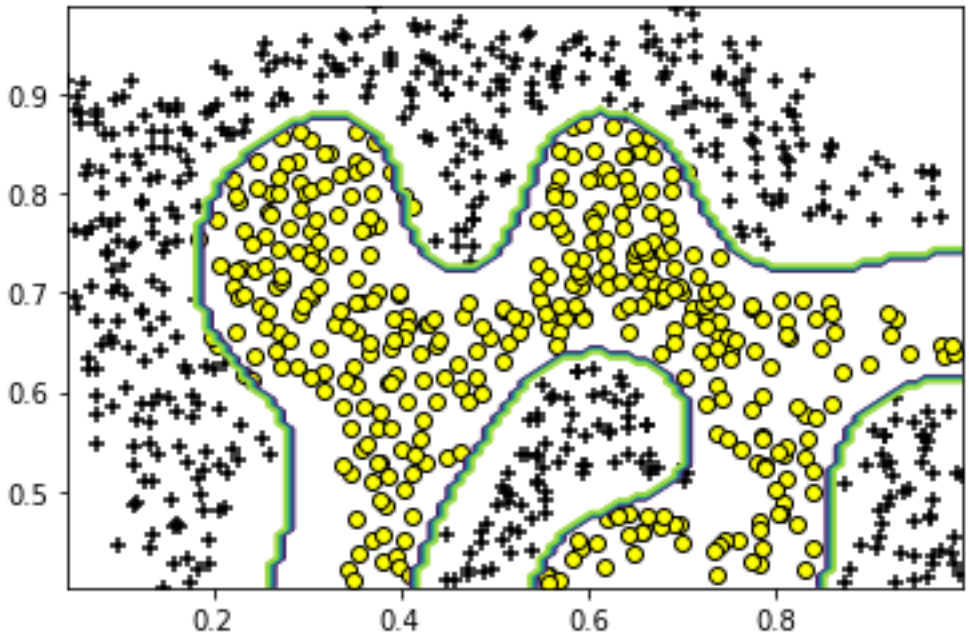


C=1



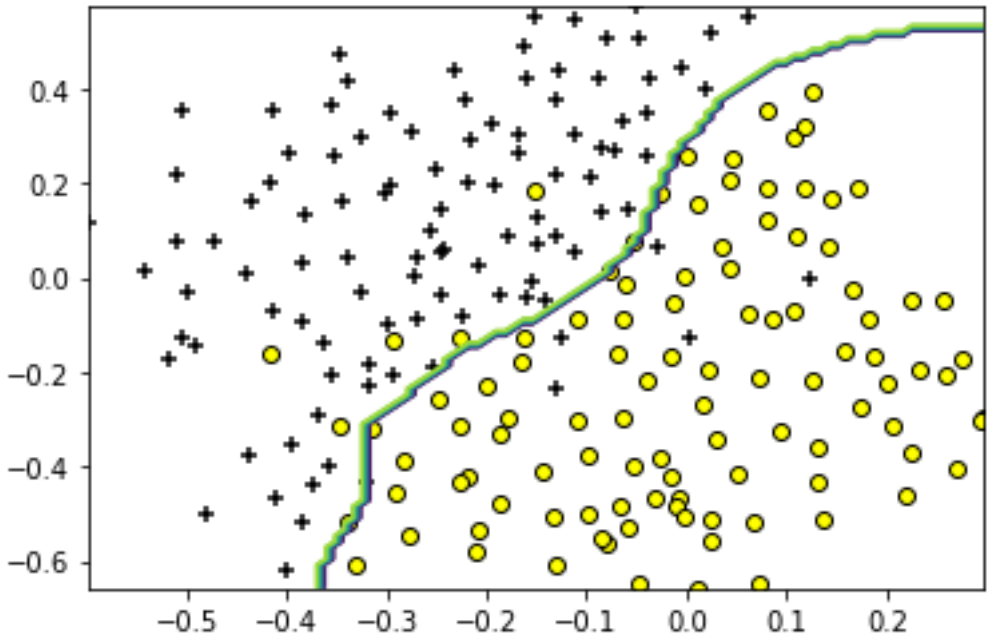
C=100

A continuación, tenemos el mismo problema, solo que el conjunto de datos no es linealmente separable. Por ello, utilizamos el kernel gaussiano para conseguir clasificar los datos.



El Kernel Gaussiano, a parte del parámetro C, hace uso de otro parámetro, sigma. Por ello, hemos creado una función que devuelve el mejor ajuste de estos parámetros, usando como valores a probar el conjunto [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]. Para saber si una elección C-Sigma es mejor que otra, se utiliza la función accuracy_score de la biblioteca sklearn.

Para el tercer ejemplo de entrenamiento, obtenemos el siguiente resultado.



Dando como mejores valores C=1 y Sigma = 0.1

```
C=1 BestSigma =0.1
```

- Código de la práctica: parte dos

```
def get_data(email):
    words = getVocabDict()
    mail = np.zeros(len(words)+1)
    for word in email:
        if (word in words):
            mail[words[word]] = 1

    return mail

#2. Detección de spam

    Xmails = []

    print("Leyendo spam")
    for i in range (1, 501):
        email_contents = codecs.open('./p6/{0}/{1:04d}.txt'.format('spam'
, i), 'r', encoding='utf-8', errors='ignore').read()
        email = email2TokenList(email_contents)
        Xmails.append(get_data(email))
    print("Leyendo easyham")
    for i in range (1, 2552):
        email_contents = codecs.open('./p6/{0}/{1:04d}.txt'.format('easy_
ham', i), 'r', encoding='utf-8', errors='ignore').read()
        email = email2TokenList(email_contents)
        Xmails.append(get_data(email))
    print("Leyendo hardham")
    for i in range (1, 251):
        email_contents = codecs.open('./p6/{0}/{1:04d}.txt'.format('hard_
ham', i), 'r', encoding='utf-8', errors='ignore').read()
        email = email2TokenList(email_contents)
        Xmails.append(get_data(email))

    Ymails = np.concatenate((np.ones(500), np.zeros(2551), np.zeros(250))
)
    Xmails, Ymails = shuffle(Xmails, Ymails, random_state=0)

    print("He randomizado los ejemplos")



    Xvalmails, Yvalmails = Xmails[:int(len(Xmails)*0.75)], Ymails[:int(le
n(Ymails)*0.75)]
    Xtrainmails, Ytrainmails = Xmails[int(len(Xmails)*0.75):], Ymails[int
(len(Ymails)*0.75):]
    C, sigma, score  = selectCandSigma(Xtrainmails, Ytrainmails, Xvalmail
s, Yvalmails)
```

```
    print ("Precision con entrenamiento (75% de los casos) y validacion(2
5% de los casos): ", score)
```

- Resultados de ejecución: parte 2

En esta segunda parte, se hace por fin un ejemplo real para clasificar correos con spam. Para ello, se leen muchos ejemplos gracias a bibliotecas otorgadas por el profesor. Tras leer y parsear los correos y adaptarlos al vector de 1s y 0s, procedemos a encontrar el mejor ajuste C y Sigma y obtener su puntuación (no hemos sido capaces de visualizar la frontera). Para ello, hemos decidido entrenar con el 75% de los mails y validar con el 25% de estos. Antes de estos, hemos 'barajado' los emails, ya que se leyeron en orden y siempre escogería los mismo ejemplos para el mismo cometido. Tras esto, obtenemos el siguiente resultado.

```
Leyendo spam
Leyendo easyham
Leyendo hardham
He randomizado los ejemplos
Precision con entrenamiento (75% de los casos) y validacion(25% de los casos):  0.9713131313131314
C=10 BestSigma =10
```

- Conclusiones

Esta práctica ha sido muy interesante, ya que hemos podido comprobar la potencia de las maquinas de vectores de soporte, haciendo un clasificador de spam que podría ser útil en un entorno real con muy pocas líneas de código. Pensábamos que tardaría más tiempo en clasificar, pero al final no fue para tanto.