Práctica 6: Support Vector Machines

Alumnos: Andrés Ruiz Bartolomé y Adrián de Lucas Gómez

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from scipy.optimize import minimize
from sklearn.svm import SVC
from process_email import email2TokenList
import codecs
from get_vocab_dict import getVocabDict
import glob
import sklearn.model_selection as ms
import time
```

1.0: SVM`s

Para esta práctica el objetivo es el aprender a usar las **Support Vector Machines** para poder dividir y catalogar distintos conjuntos que hayen los datos

Para poder visualizar las funciones usaremos **visualize_boundary**

In [9]:
```
def visualize_boundary(X, y, svm):
    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()
```

1.1: Kernel linear

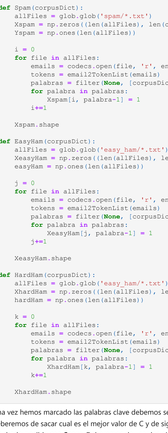
Usando los datos de **ex6data1** que contiene datos que se puden separar con una recta. A mayor sea el valor de C la recta se adapta mas a los datos. Con valor C=100 se observa que esta casi tocando con algunos de los puntos de conjuntos distintos mientras que con C=1 sería mas imparcial.

In [10]:
```
def parte1C1():
    data = loadmat('ex6data1.mat')
    X, y = data['X'], data['y'].ravel()
    svm = SVC(kernel = 'linear', C=1)
    svm.fit(X, y.ravel())
    visualize_boundary(X, y, svm)

parte1C1()

def parte1C100():
    data = loadmat('ex6data1.mat')
    X, y = data['X'], data['y'].ravel()
    svm = SVC(kernel='linear', C=100)
    svm.fit(X, y.ravel())
    visualize_boundary(X, y, svm)

parte1C100()
```

1.2: Kernel Gaussiano

Ahora usaremos el kernel gaussiano con el objetivo de separar dos conjuntos que no se pueden separar usando una recta. Los cálculos se harán sobre el conjunto de datos **ex6data2**. Usamos un sigma de 0.1 y C=1

In [11]:
```
def parte2():
    data = loadmat('ex6data2.mat')
    X, y = data['X'], data['y'].ravel()
    sigma = 0.1
    C = 1
    svm = SVC(kernel='rbf', C=C, gamma=1/(2 * sigma**2))
    svm.fit(X, y.ravel())
    visualize_boundary(X, y, svm)

parte2()
```



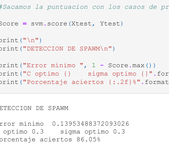
1.3: Elección de parámetros optimos C y sigma

Ahora trataremos de encontrar los valores de *C* y sigma para tratar de reducir el error. Probaremos 8 valores posibles para cada parámetro probando todas sus combinaciones.

Usando los valores de entrenamiento y validación del archivo **ex6data3** deberemos de ver el menor procentaje de error que nos da el algoritmo.

In [12]:
```
data = loadmat('ex6data3.mat')
X, y = data['X'], data['y'].ravel()
Xval, yVal = data['Xval'], data['yval'].ravel()
sigmas = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
Cs = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
scores  = np.zeros((len(Cs), len(sigmas)))

i = 0
j = 0
for c,i in zip(Cs,range(len(Cs))):
    for sigma,j in zip(sigmas,range(len(sigmas))):
        svm = SVC(kernel='rbf', C=c, gamma=1/(2 * sigma**2))
        svm.fit(X,y.ravel())
        scores[i,j] = svm.score(Xval, yVal)

print("Error minimo ", 1 - scores.max())
cOpt = Cs[scores.argmax()//len(sigmas)]
sigmaOpt = sigmas[scores.argmax() % len(sigmas)]
print("C optimo ()    sigma optimo ()".format(cOpt, sigmaOpt) )
svm = SVC(kernel='rbf', C=cOpt, gamma=1/(2 * sigmaOpt**2))
svm.fit(X,y.ravel())
visualize_boundary(X, y, svm)
```

```
Error minimo  0.035000000000000003
C optimo 1    sigma optimo 0.1
```



2.0: Detección de Spam

Para comezar con la parte de detección de SPAM en correos electrónicos primero deberemos de marcar las palabras que se leen de los correos que sabemos que son SPAM, EASY_HAM y HARD_HAM. Para eso hemos hecho un método pero para cada tipo que los lee y va marcando las palabras que tenemos en el diccionario.

In [13]:
```
def Spam(corpusDict):
    allFiles = glob.glob('spam/*.txt')
    Xspam = np.zeros((len(allFiles), len(corpusDict)))
    Ispam = np.ones(len(allFiles))

    i = 0
    for file in allFiles:
        emails = codecs.open(file, 'r', encoding='utf-8', errors = 'ignore').read()
        tokens = email2TokenList(emails)
        palabras = filter(None, [corpusDict.get(x) for x in tokens])
        for palabra in palabras:
            Xspam[i, palabra-1] = 1
        i+=1

    Xspam.shape

def EasyHam(corpusDict):
    allFiles = glob.glob('easy_ham/*.txt')
    XeasyHam = np.zeros((len(allFiles), len(corpusDict)))
    easyHam = np.ones(len(allFiles))

    j = 0
    for file in allFiles:
        emails = codecs.open(file, 'r', encoding='utf-8', errors = 'ignore').read()
        tokens = email2TokenList(emails)
        palabras = filter(None, [corpusDict.get(x) for x in tokens])
        for palabra in palabras:
            XeasyHam[j, palabra-1] = 1
        j+=1

    XeasyHam.shape

def HardHam(corpusDict):
    allFiles = glob.glob('easy_ham/*.txt')
    XhardHam = np.zeros((len(allFiles), len(corpusDict)))
    hardHam = np.ones(len(allFiles))

    k = 0
    for file in allFiles:
        emails = codecs.open(file, 'r', encoding='utf-8', errors = 'ignore').read()
        tokens = email2TokenList(emails)
        palabras = filter(None, [corpusDict.get(x) for x in tokens])
        for palabra in palabras:
            XhardHam[k, palabra-1] = 1
        k+=1

    XhardHam.shape
```

Una vez hemos marcado las palabras clave debemos separar los ejemplos en **validación, entrenamiento y prueba**. Tras separarlos deberemos de sacar cual es el mejor valor de C y de sigma (los mas precisos) y son los que pasaremos a la **SVM** para asi obtener el mejor resultado posible con **Score**. Deberemos de usar los ejemplos de prueba, es decir, los que no se han usado en el entrenamiento y validación para sacar la precisión real de la **SVM** que hemos entrenado.

In [14]:
```
# Parte 4: Deteccion de SPAM-------------------------------------------------------------------

corpusDict = getVocabDict()

#Marcamos las palabras clave de cada conjunto
Spam(corpusDict)
EasyHam(corpusDict)
HardHam(corpusDict)

Xtrain, Xtest, Ytrain, Ytest = ms.train_test_split(X, y, test_size=0.2, random_state=1)
Xtrain, Xvalid, Ytrain, Yvalid = ms.train_test_split(Xtrain, Ytrain, test_size=0.25, random_state=1)

#Sacamos los mejores valores para C y sigma

sigmas = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
Cs = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

correct = np.empty((len(Cs), len(sigmas)))

for c,i in zip(Cs,range(len(Cs))):
    for sigma,j in zip(sigmas,range(len(sigmas))):
        svm = SVC(kernel='rbf', C=c, gamma=1/(2 * sigma**2))
        svm.fit(Xtrain,Ytrain)
        correct[i,j]= svm.score(Xvalid, Yvalid)

COpt = Cs[correct.argmax()//len(Cs)]
SigmaOpt = sigmas[correct.argmax() % len(sigmas)]

svm = SVC(kernel='rbf', C=COpt, gamma=1/(2 * SigmaOpt**2))
svm.fit(Xtrain, Ytrain)

#Sacamos la puntuacion con los casos de prueba (Los no usados previamente)

Score = svm.score(Xtest, Ytest)

print("\n")
print("DETECCION DE SPAMM\n")

print("Error minimo ", 1 - Score.max())
print("C optimo ()    sigma optimo ()".format(COpt, SigmaOpt) )
print("Porcentaje aciertos {:.2f}%".format(Score*100))
```

```
DETECCION DE SPAMM

Error minimo  0.13953488372093026
C optimo 0.3    sigma optimo 0.3
Porcentaje aciertos 86.05%
```