Practica 6

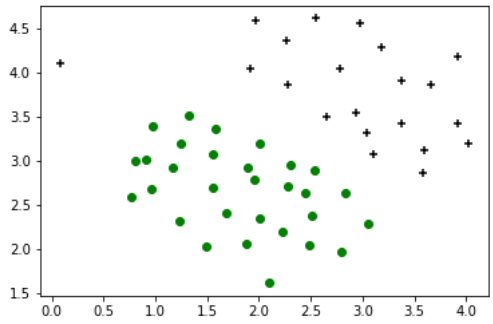
Antonio Rodríguez Hurtado, Miguel Ferreras Chumillas

SVM CON KERNEL LINEAL

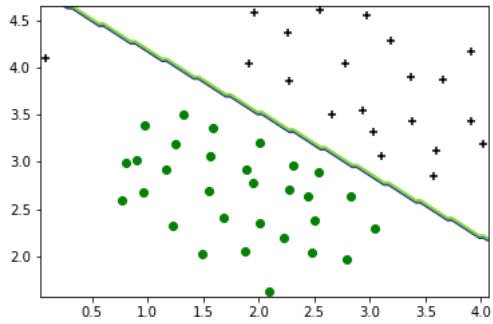
Importamos librerias, definimos las funciones de visualizacion y cargamos los datos necesarios para implementar el smv.

```
#   Librerias
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

#   Funcion de visualizacion de las graficas
def visualize_data(X, y, file_name):
    pos = np.where(y == 1.0)
    neg = np.where(y == 0.0)
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
    plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g')
    plt.savefig(file_name)
    plt.close()

def visualize_boundary(X, y, svm, file_name):
    margin = 0.05
    x1 = np.linspace(X[:, 0].min() - margin, X[:, 0].max() + margin, 100)
    x2 = np.linspace(X[:, 1].min() - margin, X[:, 1].max() + margin, 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
    pos = np.where(y == 1.0)
    neg = np.where(y == 0.0)
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], marker='+', c='k')
    plt.scatter(X[neg, 0], X[neg, 1], marker='o', c='g')
    plt.contour(x1, x2, yp)
    plt.savefig(file_name)
    plt.close()


#   Cargamos los datos del primer conjunto
data = loadmat("ex6data1.mat")
X = data["X"]
y = data["y"].ravel()

#   Visualizamos los datos del primer conjunto
visualize_data(X, y, "data_1")

#   Creamos el clasificador linal y lo entrenamos para C = 1
svm = SVC(kernel='linear', C=1.0)
svm.fit(X, y)
visualize_boundary(X, y, svm, "svm_l_c_1")

#   Creamos el clasificador lineal y lo entrenamos para C = 100
svm = SVC(kernel='linear', C=100)
svm.fit(X, y)
visualize_boundary(X, y, svm, "svm_l_c_100")
```

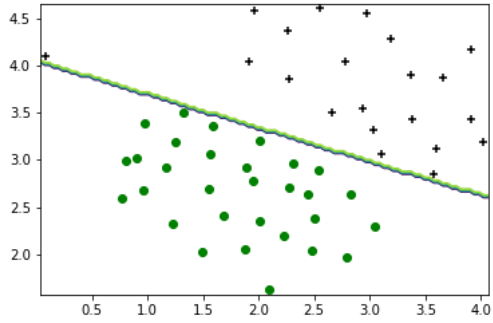
En primer lugar se visualizan los datos, como vemos se pueden separar facilmente de una forma lineal entre arriba a la derecha y abajo a la izquierda.



Probamos con C = 1 y vemos que la frontera se situa de forma muy intuitiva separando la mayoria de los datos correctamente.



Al aumentar a C = 100 vemos como la recta se sobreajusta demasiado a los datos existentes incluyendo tambien un dato muy separado del resto.



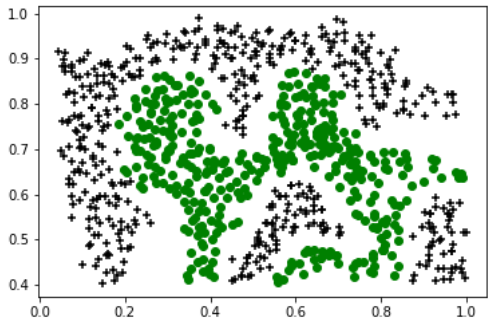
SVM CON KERNEL GAUSSIANO

En este caso trabajamos con datos que es imposible separar con una recta, para ello utilizamos el kernel gaussiano

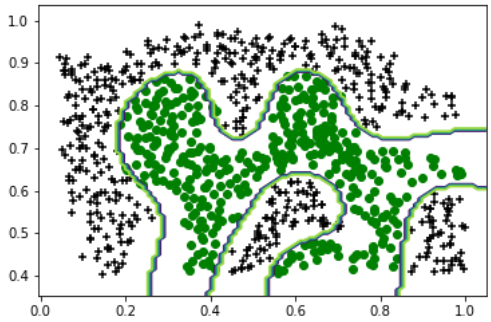
```
#   Cargamos los datos del segundo conjunto
data = loadmat("ex6data2.mat")
X = data["X"]
y = data["y"].ravel()

#   Definimos las variables que vamos a usar en el entrenamiento del svm
C = 1.0
sigma = 0.1

#   Visualizamos los datos del segundo conjunto
visualize_data(X, y, "data_2")

#   Creamos el clasificador gaussiano y lo entrenamos
```

```
svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
svm.fit(X, y)
visualize_boundary(X, y, svm, "svm_g")
```

Los datos son los siguientes:



Al aplicar el smv vemos como este se adapta a los datos para clasificarlos con una forma mucho mas compleja a una recta.

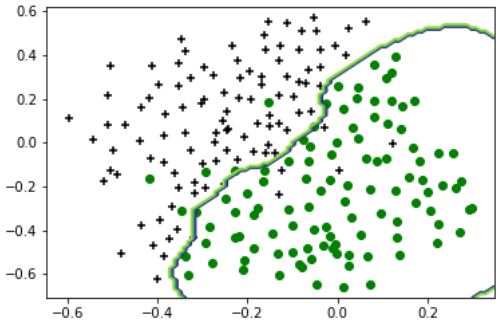


ELECCION DE LOS PARAMETROS C Y SIGMA

A continuacion probamos distintos valores de C y sigma sobre un nuevo conjunto de datos para encontrar el que nos de una mejor precision.

```
#   Cargamos los datos del tercer conjunto
data = loadmat("ex6data3.mat")
X = data["X"]
y = data["y"].ravel()
Xval = data["Xval"]
yval = data["yval"].ravel()

#   Definimos nuestro conjunto de valores para C y sigma
values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

#   Calculamos que combinacion es la mejor
best_C = 0
best_sigma = 0
best_acc = 0
for C in values:
    for sigma in values:
        svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
        svm.fit(X, y)
        acc = accuracy_score(yval, svm.predict(Xval))
        if acc >= best_acc:
            best_C = C
            best_sigma = sigma
            best_acc = acc

print("Mejor C: ", best_C, ", Mejor sigma: ", best_sigma, ", Mejor precision: ", best_acc)
svm = SVC(kernel='rbf', C=best_C, gamma=1 / (2 * best_sigma**2))
svm.fit(X, y)
visualize_boundary(X, y, svm, "svm_g_best")
```

Este el el resultado, que se adapta a los datos con un precision de 0.965 y vemos como no se sobreajusta en exceso permitiendo a datos no representativos salirse de la clasificacion. Este resultado se consigue con C = 3 y sigma = 0.1



Parte 2: Deteccion de Spam

Importamos las librerias y contruimos las funciones necesarias para crear los datos

```
#   Librerias
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from codecs import open
import get_vocab_dict as vodi
import process_email as poem
from sklearn.model_selection import train_test_split


#   Numero de correos por carpeta
n_spam = 500
n_easy_ham = 2551
n_hard_ham = 250

dic = vodi.getVocabDict()
n_words = len(dic)

def parse_name(number):
    while(len(str(number)) < 4):
          number = '0' + str(number)
    return str(number)

def create_X(n_emails, name_folder):
    X = np.array
    for i in range(1, n_emails + 1):
        email_contents = open(name_folder+'/'+parse_name(i)+'.txt', 'r', encoding='utf-8', errors='ignore').read()
        email = poem.email2TokenList(email_contents)
        Xn = np.zeros(n_words)
        for word in email:
            if word in dic:
                Xn[dic[word] - 1] = 1
        if i == 1:
            X = Xn
        else:
            X = np.vstack([X, Xn])
    return X
```

Seguidamente sacamos los datos de cada carpeta y los separamos en dos conjuntos con una relacion de 0.8/0.2 para entrenamiento y test respectivamente. Consideramos la y como un 1 si el correo es spam, y como un 0 si no lo es.

```
X_train, X_test, y_train, y_test = train_test_split( create_X(n_spam, 'spam'), np.ones(n_spam), test_size=0.2, random_state=0)
X_train_aux, X_test_aux, y_train_aux, y_test_aux = train_test_split( create_X(n_easy_ham, "easy_ham"), np.zeros(n_easy_ham), test_size

#Unimos los datos de la carpeta a los anteriores
X_train = np.vstack([X_train, X_train_aux])
X_test = np.vstack([X_test, X_test_aux])
```

```
y_train = np.append(y_train, y_train_aux)
y_test = np.append(y_test, y_test_aux)

#repetimos con la ultima carpeta
X_train_aux, X_test_aux, y_train_aux, y_test_aux = train_test_split( create_X(n_hard_ham, "hard_ham"), np.zeros(n_hard_ham), test_size
X_train = np.vstack([X_train, X_train_aux])
X_test = np.vstack([X_test, X_test_aux])
y_train = np.append(y_train, y_train_aux)
y_test = np.append(y_test, y_test_aux)
```

Seguidamente repetimos el proceso de la parte 1 de la practica para hallar la mejor C y la mejor sigma:

```
values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
best_C = 0
best_sigma = 0
best_acc = 0
for C in values:
    for sigma in values:
        svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
        svm.fit(X_train, y_train)
        acc = accuracy_score(y_test, svm.predict(X_test))
        print(C,sigma, "acuraccy: ", acc)
        if acc >= best_acc:
            best_C = C
            best_sigma = sigma
            best_acc = acc

print(best_acc, best_C, best_sigma)
```

Encontramos que la mejor combinacion es C = 30 y sigma = 1 donde nos da una precision de 0.986