***Práctica 6***

***Support Vector Machines***

Esta práctica 6 se divide en dos partes bien diferenciadas. La primera consiste en adaptar y familiarizarse con el uso del clasificador SVM que después aplicaremos en la segunda parte para implementar un clasificador de SPAM.

Implementaremos una versión que nos permitirá clasificar bases de datos linealmente separables (primeara parte que hace referencia al uso del Kernel lineal), y una versión que mejoraremos en el tercer apartado de la primera parte y que nos permitirá separar bases de datos que no son linealmente separables y que nos obligarán a usar el Kernel gaussiano o un equivalente como podría ser el RBF, sin embargo, nosotros utilizaremos la opción “precomputed “, que nos permite aproximarnos a los valores buscados a la hora de obtener la SVM mediante Kernel gausiano.

A lo largo de la práctica utilizaremos diversas funciones auxiliares que nos permitirán, entre otras cosas, cargar la base de datos para poder procesarla, así como representar los resultados obtenidos de manera gráfica.

Kernel lineal:

Para la primera parte se nos pide implementar una SVM mediante Kernel lineal y que experimentemos con las diferencias que supone variar el parámetro C de regularización y bajo unos valores de tol y un máximo de iteraciones fijos.

Probaremos primero con un valor de C = 1 y posteriormente realizaremos el análisis con un valor de C = 100.

El método principal que engloba la lógica necesaria para los cálculos pedidos tiene el siguiente aspecto:

def part1\_main():

    #Parte 1.1

    X, y, y\_r = load\_data("ex6data1.mat")

    # C = 1

    c\_param = 1

    svm\_function = SVM\_linear\_training(X, y\_r, c\_param)

    draw\_Linear\_KernerFrontier(X, y\_r, svm\_function)

    # C = 100

    c\_param = 100

    svm\_function = SVM\_linear\_training(X, y\_r, c\_param)

    draw\_Linear\_KernerFrontier(X, y\_r, svm\_function)

El el se lanza el método que nos permite vaciar la base de datos en los vectores correspondientes, el método que nos calcula la función SVM aplicando Kernel gaussiano con los parámetros dados y el método auxiliar que nos permite dibujar la función SVM y representarla gráficamente.

Empezaremos cargando los valores de nuestra base de datos en los vectores correspondientes gracias al método que llamamos load data:

def load\_data(file\_name):

    data = loadmat(file\_name)

    X = data['X']

    y = data['y']

    y\_r = np.ravel(y)

    return X, y, y\_r

Para el dibujado de los elementos de la base de datos y el resultado de nuestro clasificador utilizaremos la función *draw\_Linear\_KernerFrontier*, que recibe los vectores de parámetros y resultado y la función SVM calculada gracias a nuestro método *SVM\_linear\_training*.

El primero tiene el siguiente aspecto:

def draw\_Linear\_KernerFrontier(X, y, svm\_function):

    w = svm\_function.coef\_[0]

    a = -w[0] / w[1]

    #seleccionamos dos puntos de la recta para representarla

    p1 = np.array([X[:,0].min(), X[:,0].max()])

    p2 = a \* p1 - (svm\_function.intercept\_[0]) / w[1]

    #Frontera de separación

    plt.plot(p1, p2, c = 'y')

    displayData(X, y)

    plt.show()

La función de mayor importancia y que representa el entrenamiento de nuestro clasificador recibe como parámetros de entrada nuevamente los vectores con los datos de entrenamiento y resultados, así como el valor C de regularización del que depende la precisión de nuestra clasificación:

def SVM\_linear\_training(X, y, c\_param):

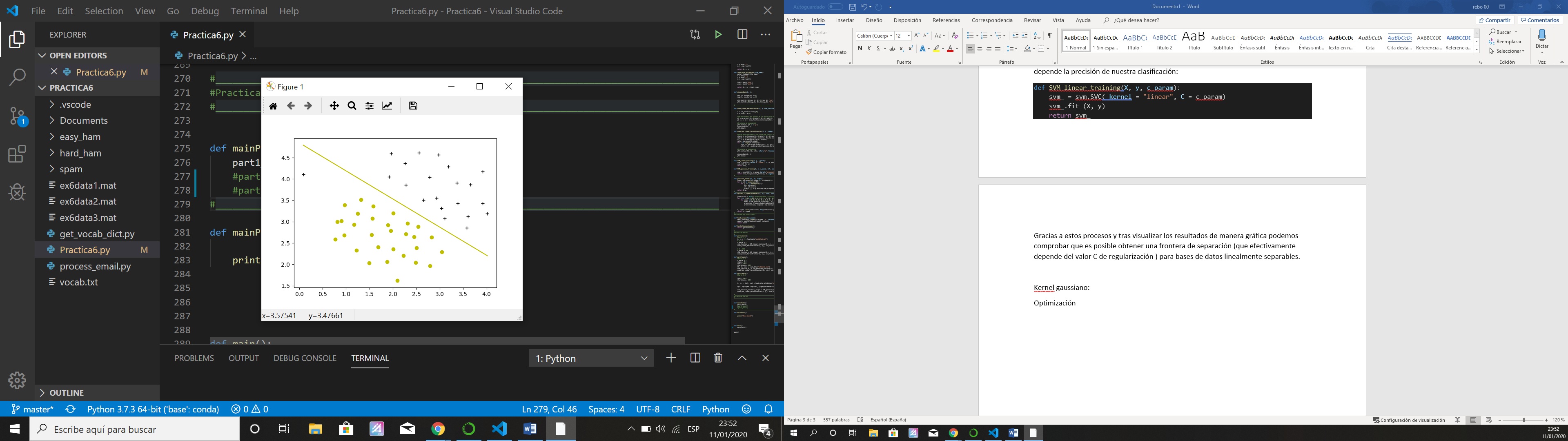
    svm\_ = svm.SVC( kernel = "linear", C = c\_param)

    svm\_.fit (X, y)

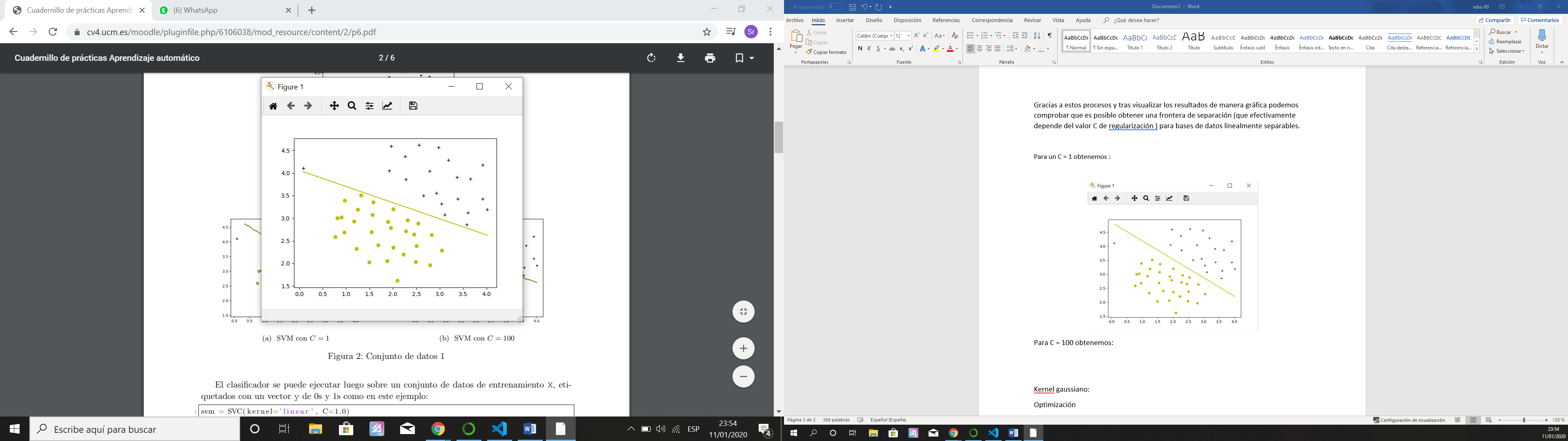
    return svm\_

Gracias a estos procesos y tras visualizar los resultados de manera gráfica podemos comprobar que es posible obtener una frontera de separación (que efectivamente depende del valor C de regularización ) para bases de datos linealmente separables.

Para un C = 1 obtenemos :



Para C = 100 obtenemos:



Podemos apreciar claramente que para C = 100 todos los valores quedan separados correctamente, dándonos a entender que, valores más cercanos en este caso a 100, van a ser más precisos que aquellos más bajos y próximos al 1. Podríamos concluir que valores razonablemente más altos conllevan una mayor precisión a la hora de calcular la frontera de separación de bases de datos con valores linealmente separables.

Kernel gaussiano:

En este apartado tendremos que entrenar nuestra SVM mediante el uso de un Kernel gausiano para poder clasificar un conjunto de valores no separables linealmente.

Nuestra función principal tiene este aspecto:

def part2\_main():

    #Parte 1.2

    c\_param = 1

    sigma = 0.1

    tool = 1e-3

    iterations = 100

    X1, y1, y1\_r = load\_data("ex6data2.mat")

    svm\_function\_n\_l = SVM\_gaussian\_training(X1, y1\_r, c\_param, tool, iterations, sigma)

    draw\_Non\_Linear\_KernelFrontier(X1, y1\_r, svm\_function\_n\_l, sigma)

Nuevamente hacemos uso del método de volcado de la base de datos de entrenamiento.

Nuestra función que nos permite pintar en pantalla el resultado ha sido adaptada y ha cambiado radicalmente su aspecto, pues ya no trabaja con una función lineal:

def draw\_Non\_Linear\_KernelFrontier(X, y , model, sigma):

    #Datos que conformarán la curva que servirá de frontera

    x1plot = np.linspace(X[:,0].min(), X[:,0].max(), 100).T

    x2plot = np.linspace(X[:,1].min(), X[:,1].max(), 100).T

    X1, X2 = np.meshgrid(x1plot, x2plot)

    vals = np.zeros(X1.shape)

    for i in range(X1.shape[1]):

        this\_X = np.column\_stack((X1[:, i], X2[:, i]))

        vals[:, i] = model.predict(gaussian\_Kernel(this\_X, X, sigma))

    #Frontera de separación

    plt.contour(X1, X2, vals, colors="y", linewidths = 0.1 )

    displayData(X, y)

    plt.show()

En este caso no solo debe recibir como parámetro de entrada los datos de entrenamiento y la SVM correspondiente , sino que para su cálculo también necesita los valores de sigma para el cálculo de la frontera de separación correspondiente.

Nuestra función principal SVM esta vez tiene dependencia directa no solo del parámetro C de regularización si no del parámetro tol y del número de iteraciones y que afectan de manera directa a la precisión en el cálculo de la frontera de división o en la correcta clasificación de los valores de entrenamiento.

def SVM\_gaussian\_training(X, y, c\_param, tol, max\_i, sigma):

    svm\_ = svm.SVC(C = c\_param, kernel="precomputed", tol = tol, max\_iter = max\_i)

    return svm\_.fit(gaussian\_Kernel(X, X, sigma=sigma), y)

Como ya ocurría a la hora de dibujar la frontera de separación o clasificación necesitamos hacer uso de la función que utilizamos como base para estos cálculos: La función que nos permite calcular el Kernel Gaussiano:

def gaussian\_Kernel(X1, X2, sigma):

    Gram = np.zeros((X1.shape[0], X2.shape[0]))

    for i, x1 in enumerate(X1):

        for j, x2 in enumerate(X2):

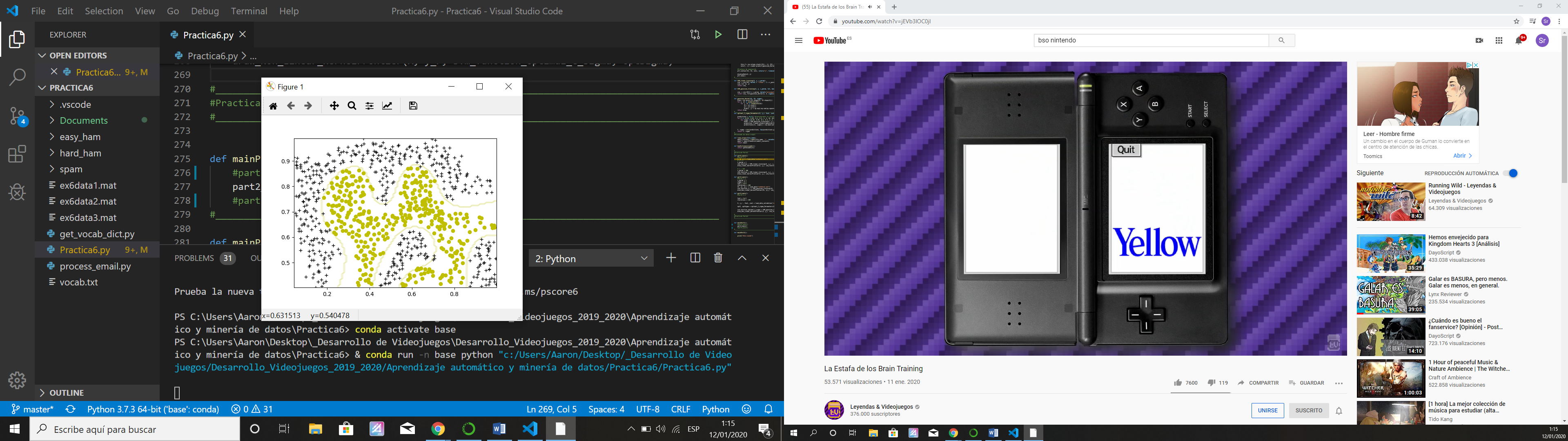
            x1 = x1.ravel()

            x2 = x2.ravel()

            Gram[i, j] = np.exp(-np.sum(np.square(x1 - x2)) / (2 \* (sigma\*\*2)))

    return Gram

Tras lanzar el programa el resultado obtenido es el siguiente :



Como podemos comprobar la frontera clasifica con bastante precisión los datos de nuestra base y el resultado obtenido es similar al sugerido por el guion de la práctica sobre la que estamos trabajando.

Optimización:

Esta tercera parte nos invita a elaborar un sistema de SVM que tenga en cuenta los parámetros que ya teníamos en cuenta hasta ahora, además de un conjunto de valores de validación y que nos permite trabajar sobre porcentajes de clasificación correcta, así como obtener los valores de C y sigma óptimos o que acarrean clasificaciones más favorables o con porcentajes de acierto en la clasificación más altos.

El método de carga de valores ha cambiado ligeramente, pues ahora debe volcar también los datos de validación Xval e yval:

def load\_data\_validation(file\_name):

    data = loadmat(file\_name)

    X = data['X']

    y = data['y']

    y\_r = np.ravel(y)

    Xval = data['Xval']

    yval = data['yval']

    return X, y\_r , Xval, yval

La función principal en este caso tiene el siguiente aspecto:

def part3\_main():

    #Parte 1.3

    tool = 1e-3

    iterations = 100

    X, y\_r , Xval, yval = load\_data\_validation("ex6data3.mat")

    optC, optSigma = optimal\_C\_sigma\_Parameters(X, y\_r, Xval, yval, iterations, tool)

    svm\_function\_optimal\_C\_sigma = SVM\_gaussian\_training(X, y\_r, optC, tool, iterations, optSigma)

    draw\_Non\_Linear\_KernelFrontier(X, y\_r, svm\_function\_optimal\_C\_sigma, optSigma)

Como podemos ver en este caso hacemos los cálculos previos al cálculo de la SVM y su dibujado de los C y sigma óptimos mediante la función Optimal\_c\_sigma que recibe como parámetros los datos de entrenamiento y validación así como el número de iteraciones y el tol preestablecido y que tiene el siguiente aspecto :

def optimal\_C\_sigma\_Parameters(X, y\_r, Xval, yval, max\_i, tool ):

    predictions = dict() #almacenaremos la infrmacion relevante en un diccionario

    for C in [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]:

        for sigma in[0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]:

            model = SVM\_gaussian\_training(X, y\_r, C, tool, max\_i, sigma )

            prediction = model.predict(gaussian\_Kernel( Xval, X, sigma))

            predictions[(C, sigma)] = np.mean((prediction != yval).astype(int))

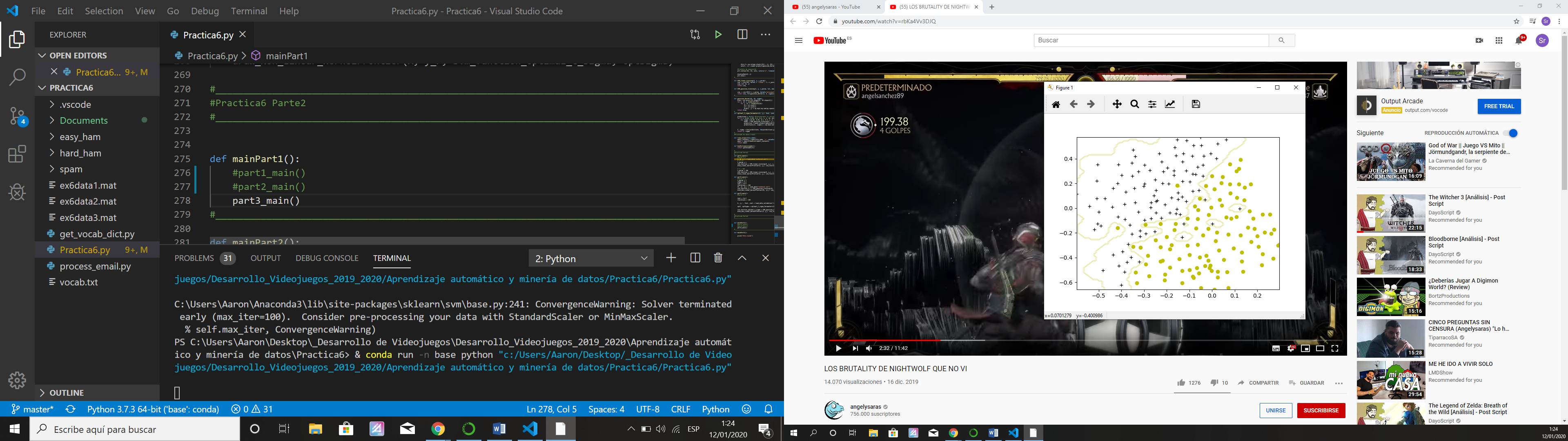
    C, sigma = min(predictions, key=predictions.get)

    return C, sigma

Para el cálculo de los valores óptimos vamos iterando con diferentes valores preestablecidos de C y sigma y mediante el uso de las funciones anteriormente utilizadas que nos dan el SVM\_gaussiano y nos permiten hallar el valor del Kernel gaussiano extrapolamos el valor de esos valores que nos darían el mayor porcentaje de acierto.

Utilizando esos valores como parámetro de entrada para nuestro entrenamiento en SVM Kernel gaussiano obtenemos la frontera de división o de clasificación óptima o con mayor tasa de acierto. Para representarlo gráficamente utilizamos nuevamente la función auxiliar que nos permite dibujar las fronteras no lineales *draw\_Non\_Linear\_KernelFrontier*.

El resultado obtenido es el siguiente:



Si bien es cierto que el resultado difiere de el sugerido por el guion de la práctica la frontera dibujada parece separar claramente y de manera bastante precisa el conjunto de datos de entrenamiento lo que nos hace tener una percepción optimista de los cálculos realizados y de la solución implementada.