Javier Cordero Calvo y Juan Ruiz Jiménez

Aprendizaje Automático y Minería de Datos – Práctica 6

1. Support Vector Machines:

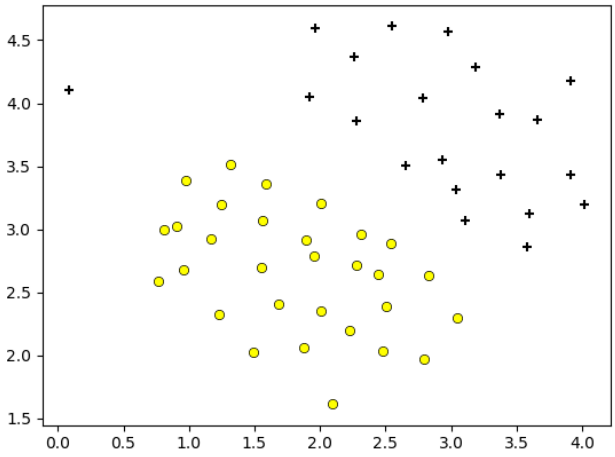
1.1 Kernel lineal:

Para la primera parte, leeremos la entrada de “ex6data1.mat” y la guardaremos en la variable data.

Tras esto, asignaremos a las variables X e y sus correspondientes valores y transformaremos y para que sea un vector de 1xM.

Una vez realizada esta conversión, definiremos el coeficiente para nuestra SVM a 1.

Antes de proseguir, debemos asegurarnos que los datos de entrada son linealmente separables, por lo que vamos a representarlos para comprobarlo.




```
data = loadmat("ex6data1.mat")

X = data["X"]
y = data["y"]
yravel = np.ravel(y)

Coef = 1.0
```

Tras haber leído y ajustado la entrada, crearemos nuestra SVM, con un kernel de tipo lineal y con el coeficiente antes definido.

Después, mediante la función fit del SVM ya declarado, ajustaremos los ejemplos de X e y.

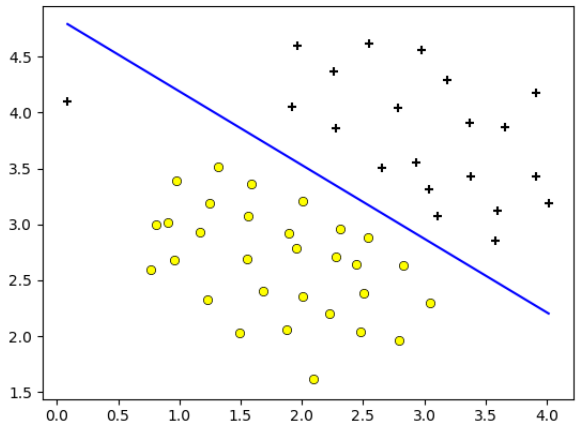
```
svm = supportVectorMachine.SVC(kernel="linear", C=Coef)
svm = svm.fit(X, yravel)
```

Tras tener nuestra SVM creada, sólo nos quedará pintar los puntos de entrada del enunciado así como la recta generada por nuestra SVM.

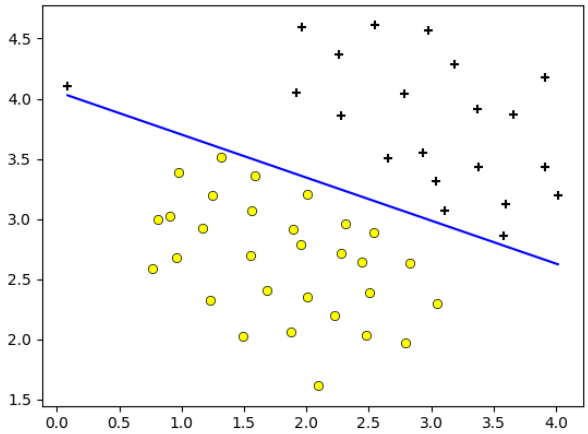
```
pinta_puntos(X, yravel)
pinta_frontera_recta(X,yravel, svm)
```

```
def pinta_puntos(X,y):
    pos = np.where(y == 1)
    plt.scatter(X[pos, 0], X[pos, 1], marker = '+', c = 'black')
    neg = np.where(y == 0)
    plt.scatter(X[neg, 0], X[neg, 1], marker = 'o', c = 'black', s = 30
)
    plt.scatter(X[neg, 0], X[neg, 1], marker = 'o', c = 'yellow', s = 2
0)
def pinta_frontera_recta(X, Y, model):
    w = model.coef_[0]
    a = -w[0] / w[1]

    xx = np.array([X[:,0].min(), X[:,0].max()])
    yy = a * xx - (model.intercept_[0]) / w[1]

    plt.plot(xx, yy, color='blue')
```

El resultado obtenido es una separación entre los conjuntos de datos no del todo precisa:



Sin embargo, cambiado simplemente el valor del coeficiente de 1 a 100, comprobamos que la recta que separa ambos conjuntos de datos se ajusta más al ejemplo esperado.



1.2 Kernel gaussiano:

Para esta parte, el primer paso a dar es leer los nuevos datos de la entrada, en este caso “ex6data2.mat” y guardarlos igual que en el ejemplo anterior, aunque esta vez definiendo un sigma = 0.1

Una vez hecho esto, generaremos el SVM esta vez sobre un kernel gaussiano (“precomputed”), y a la hora de realizar la opción de “fit” llamaremos a una función nuestra llamada gaussianKernel como primer parámetro, e y como segundo parámetro.

```
svm = supportVectorMachine.SVC(C=Coef, kernel='precomputed',
      tol= 1e-3, max_iter= 100)
svm = svm.fit(gaussianKernel(X, X, sigma=sigma), yravel)
```

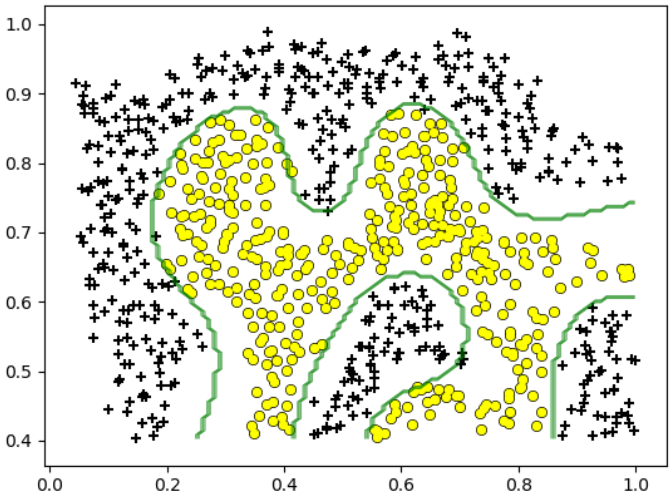
```
def gaussianKernel(X1, X2, sigma):
    Gram = np.zeros((X1.shape[0], X2.shape[0]))
    for i, x1 in enumerate(X1):
        for j, x2 in enumerate(X2):
            x1 = x1.ravel()
            x2 = x2.ravel()
            Gram[i, j] = np.exp(-
np.sum(np.square(x1 - x2)) / (2 * (sigma**2)))
    return Gram
```

Esta función realiza la operación propuesta en el enunciado de la práctica para operar con kernel gaussiano.

Finalmente, mostraremos por pantalla la curva generada por esta SVM, junto con sigma, X e y.

```
def pinta_frontera_curva(X, y, model, sigma):

    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(gaussianKernel(this_X, X,
        sigma))

    plt.contour(X1, X2, vals, colors="green", linewidths=0.3)
```

Como resultado obtenemos una gráfica en la que fácilmente se aprecia que la separación entre un conjunto de datos y otro es bastante exacta:



1.3 Elección de los parámetros de C y σ:

Para finalizar, cargaremos los datos de “ex6data3.mat” y evaluaremos el error mínimo cometido con los distintos sigmas y coeficientes que nos propone el enunciado probar.

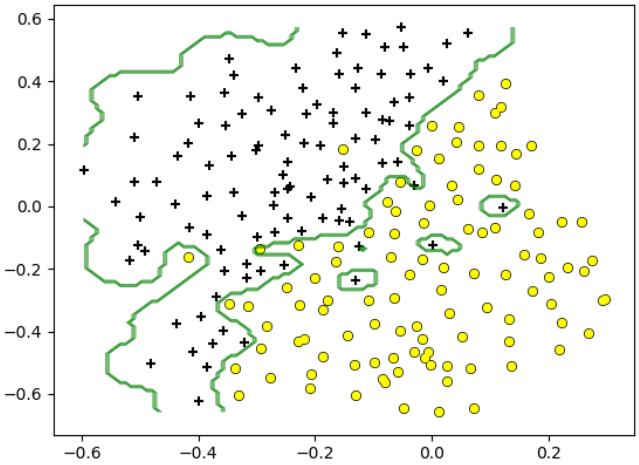
```
for x in range(8):
        for j in range(8):
            #Entrena el modelo para x e y
            model = supportVectorMachine.SVC(C=coefVal, kernel='precomputed',
                    tol= 1e-3, max_iter= 100)
            model = model.fit(gaussianKernel(X, X, sigma=sigmaVal), yravel)

            prediction = model.predict(gaussianKernel(Xval, X, sigmaVal))

            predictions[(coefVal, sigmaVal)] = np.mean((prediction != yval)
            .astype(int))

            sigmaVal = sigmaVal * 3

        sigmaVal = 0.01
        coefVal = coefVal * 3

    Coef, sigma = min(predictions, key=predictions.get)
```

Tras realizer todas las iteraciones y tener un vector con diferentes predicciones, cogemos los valores del coeficiente y de sigma que hagan la menor predicción, en este caso aproximadamente Coef = 1 y sigma = 0.03

Tras tener estos datos, simplemente entrenamos la SVM al igual que en el ejemplo anterior con el kernel gaussiano ya que los datos no se pueden separar mediante una recta.

```
svm = supportVectorMachine.SVC(C=Coef, kernel='precomputed',
      tol= 1e-3, max_iter= 100)
svm = svm.fit(gaussianKernel(X, X, sigma=sigma), yravel)

pinta_frontera_curva(X, y, svm, sigma)
```

El resultado final no es solo la curva que debería separar ambos conjuntos de datos sino la curva que engloba y separa a uno de los dos grupos de datos con respecto al otro.



2. Detección de spam:

Para esta parte de la práctica, el primer paso es leer la entrada tanto de spam como de no_spam.

Para ello, se ha creado una función llamada “read_file()” que recibe una ruta donde se contengan los archivos que se desean leer y procesar.

```
def read_file(path):
    print("Leyendo archivos de ", path)
    files = []
    for r, d, f in os.walk(path):
        for file in f:
            if '.txt' in file:
                files.append(os.path.join(r, file))
    j = 0
    vocab_dict = vocab.getVocabDict()
    arrayDict = np.array(list(vocab_dict.items()))
    X = np.zeros((len(files), len(arrayDict)))
    for f in files:
        email_contents =codecs.open(f, "r", encoding="utf-8", errors="ignore")
            .read()
        email = process_email.email2TokenList(email_contents)
        aux = np.zeros(len(arrayDict))
        for i in range(len(email)):
            index = np.where(arrayDict[:, 0] == email[i])
            aux[index] = 1
        X[j] = aux
        j = j + 1
    print("Archivos de ", path, "leídos y guardados en X.")
    return X
```

En ella, simplemente creamos una matriz del tamaño de la entrada y del número de palabras en el diccionario ya leído también, lleno de 0's.
Una vez creado y leída la entrada, pasamos a procesarla.

Para ello, por cada archivo leído, comprobamos si alguna de las palabras que contiene se encuentra en el diccionario. Si es así, la posición correspondiente en la nueva matriz creada la marcamos como 1 en lugar de 0.

Por lo tanto, al terminar tenemos una matriz de tamaño número de emails x número de palabras en el diccionario, donde un 0 implica que la palabra X[m,n] no se encuentra en el mensaje y un 1 si dicha palabra si se encuentra en el email.

Guardamos por lo tanto en X el resultado de leer la carpeta Spam.
En y guardamos un array de tamaño X lleno de 1's, ya que la salida es 1 si el mensaje contiene spam y estos mails leídos son todos de spam.

```
X = read_file('spam/')
y = np.ones(len(X)) #Los que sabemos que son spam los marcamos como
 1 en la salida
```

Acto seguido, creamos una X auxiliar, llamada XnoSpam, donde guardaremos los datos de los correos que sabemos de manera fácil que no son spam.

Creamos un vector llamado yNoSpam, de longitud XnoSpam, lleno de 0's ya que estos correos no presentan spam.

Finalmente, si juntamos X y XnoSpam, y e yNoSpam, tendremos los datos de entrenamiento de nuestra red, cuya entrada son diccionarios de palabras en los que un 0 representa que no está dicha palabra en el correo y un 1 que sí lo está y cuya salida es un vector en el cual un 1 representa que es spam y un 0 que no lo es.

```
#Juntamos los que son spam y los que no en las X's e Y's
X = np.concatenate((X, XnoSpam))
y = np.concatenate((y, yNoSpam))
```

Si entrenamos ahora la red con SVM y después comprobamos la salida para ver si coincide con los datos de entrenamiento, encontraremos la tasa de aciertos que tiene la red.

```
C = 0.1
yravel = y.ravel()
svc = svm.SVC(C, 'linear')
svc.fit(X, yravel)
p = svc.predict(X)
print('Encontramos que la red tiene una tasa de acierto del:
{0:.2f}%'.format(np.mean((p == yravel).astype(int)) * 100))
```

Tras ejecutarlo: "**Encontramos que la red tiene una tasa de acierto del: 99.84%**", el cual es un resultado bastante prometedor de que la lectura ha sido correcta y funciona.

Para comprobar completamente la funcionalidad, ahora a esta entrada X e y añadiremos los ejemplos difíciles de detectar como no spam. Si el sistema funciona, la tasa de acierto de la red debería bajar ya que para el sistema ha sido más difícil determinar si algunos ejemplos eran spam o no.

```
xnoHardSpam = read_file('hard_ham/')
ynoHardSpam = np.zeros(len(xnoHardSpam))

X = np.concatenate((X, xnoHardSpam))
y = np.concatenate((y, ynoHardSpam))

C = 0.1
yravel = y.ravel()
svc = svm.SVC(C, 'linear')
svc.fit(X, yravel)
p = svc.predict(X)

print('Encontramos que la red tiene una tasa de acierto final del:
{0:.2f}%'.format(np.mean((p == yravel).astype(int)) * 100))
```

Tras ejecutarlo: "**Encontramos que la red tiene una tasa de acierto final del: 99.73%**"

Como podemos comprobar, la tasa de aciertos ha bajado debido a la inserción de los nuevos ejemplos de entrenamiento más complejos.