Práctica 6: Support Vector Machines

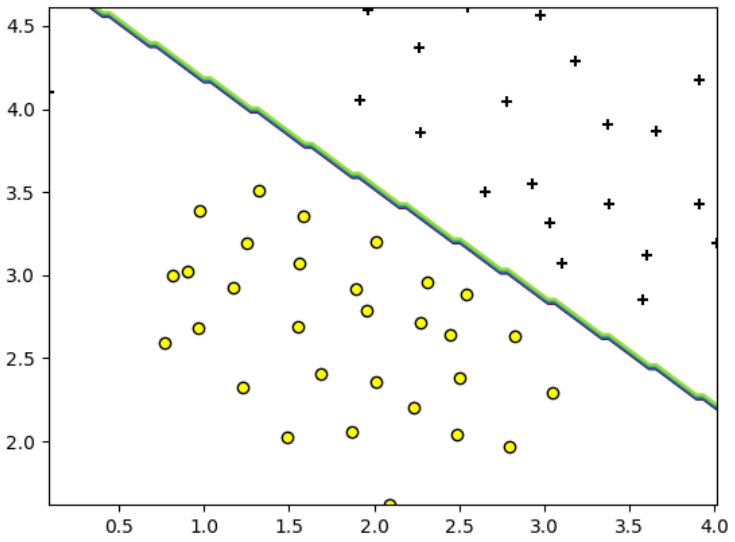
David Godoy Ruiz
Eva Lucas Leiro

Support Vector Machines:

- Kernel lineal

Utilizamos el clasificador SVM, con el parámetro C de regularización aplicando una función de kernel sobre un conjunto.

```
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.savefig(file_name)
    plt.close()
#--------------------------------------------------------------------------
X, y = loadData('ex6data1.mat')
svm = SVC(kernel= 'linear' , C=1.0)
svm.fit(X, y)
visualize_boundary(X, y,svm ,'data1_1.png')
```

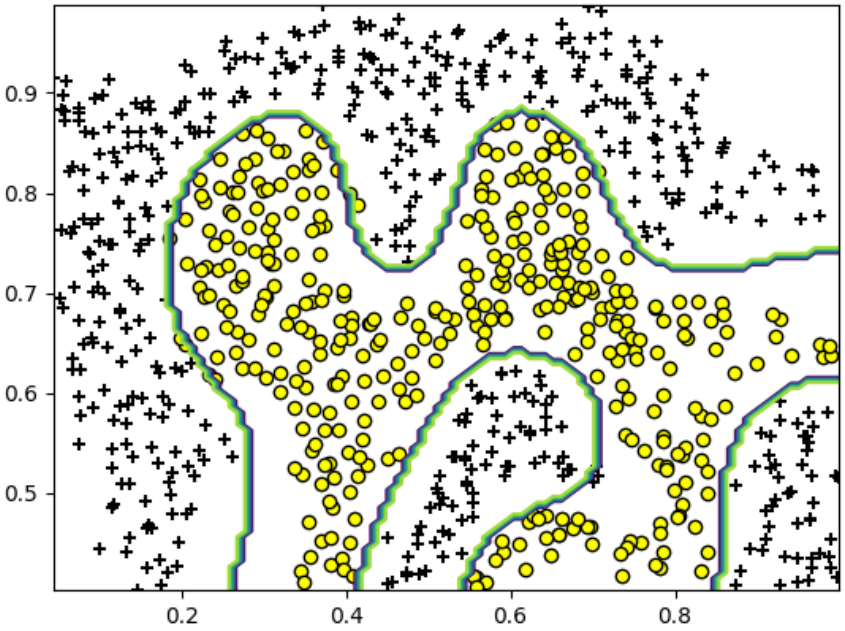


Al ser los datos linealmente separables obtenemos esta gráfica (para C = 1).

- Kernel gaussiano

Para este segundo conjunto de datos no linealmente separables debemos utilizar el kernel gaussiano.

```
X, y = loadData('ex6data2.mat')
C = 1
sigma = 0.1
svm = SVC(kernel= 'rbf' , C=C, gamma = 1/(2*sigma**2))
svm.fit(X, y)
visualize_boundary(X, y,svm ,'data1_2.png')
```



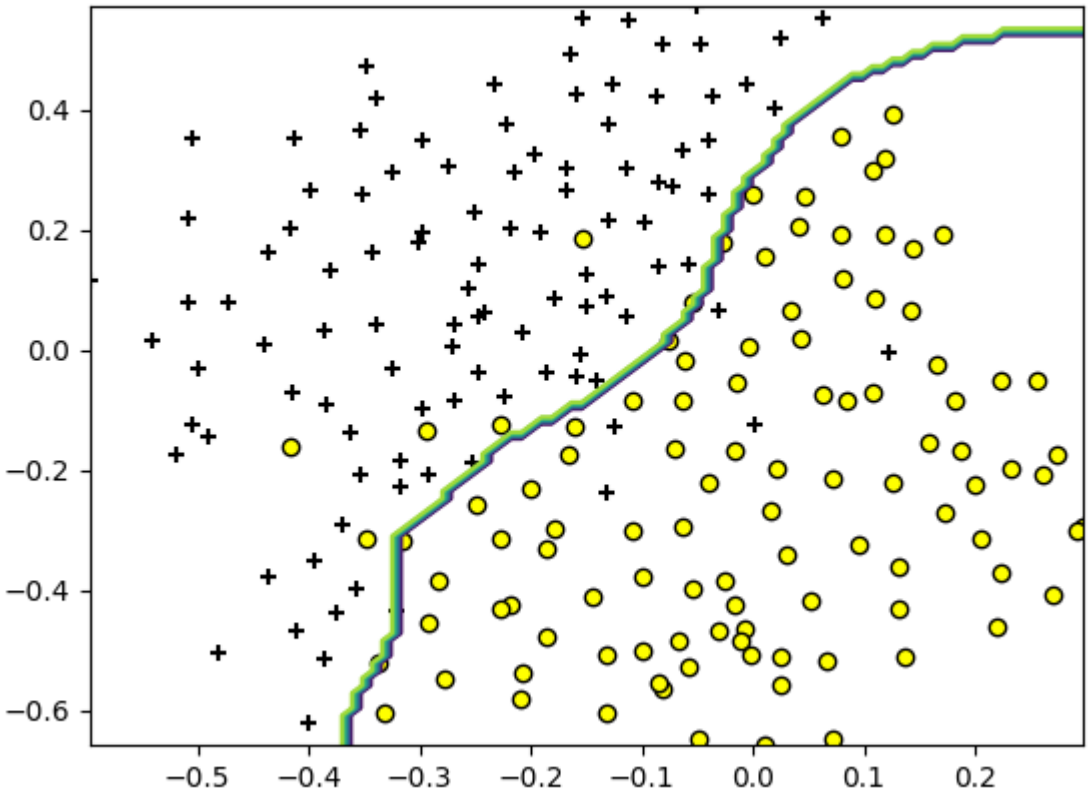
- Elección de los parámetros C y σ

Definimos un método para elegir los parámetros C y σ tomando valores del conjunto definido en el enunciado.

```
def calculaScore(svm, Xval, yval):
    yp = svm.predict(Xval).reshape(yval.shape)
    return sum(yp == yval)

#---------------------------------------------------------------------------

def eleccionParams(X, y, Xval, yval):
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    maxScore = 0

    for i in range(len(C_vec)):
        for j in range(len(sigma_vec)):
            svm = SVC(kernel= 'rbf' , C=C_vec[i], gamma = 1/(2*sigma_vec[j]**2))
            svm.fit(X, y)
            score = calculaScore(svm, Xval, yval)
```

```
            if (score > maxScore):
                maxSvm = svm
                maxC = C_vec[i]
                maxSigma = sigma_vec[j]
                maxScore = score

    return maxSvm, maxScore, maxC, maxSigma
```

```
X, y, Xval, yval = loadData3()
svm, _, _, _ = eleccionParams(X, y, Xval, yval)
visualize_boundary(X, y, svm ,'data1_3.png')
```



Detección de spam:

Cargamos los datos de los emails, los convertimos en un vector de palabras y más adelante en el vector pedido de ceros y unos, comparándolo con `vocab`

```
def cargaEmails(directorio, nFiles):
    vocab = getVocabDict()
    emails = np.zeros((nFiles, len(vocab)))
    for i in range(1, nFiles+1):
        email_contents = codecs.open('{0}/{1:04d}.txt'.format(directorio, i),
'r', encoding='utf-8', errors='ignore').read()
```

```
        words = process_email.email2TokenList(email_contents)
        vec = np.zeros(len(vocab))
        for w in words:
            if w in vocab:
                vec[vocab[w]-1] = 1
        emails[i-1] = vec
    return emails

#-----------------------------------------------------------------------

def getMatColPercent(mat, ini, fin):
    return mat[(ini*mat.shape[0]).__int__():(fin*mat.shape[0]).__int__(), :]

#-----------------------------------------------------------------------

def getMatPercent(mat, ini, fin):
    return mat[(ini*mat.shape[0]).__int__():(fin*mat.shape[0]).__int__()]

#-----------------------------------------------------------------------

def parte2():
    spam = cargaEmails("spam", 500)
    y_spam = np.ones(spam.shape[0])
    easy_ham = cargaEmails("easy_ham", 2551)
    y_easy = np.zeros(easy_ham.shape[0])
    hard_ham = cargaEmails("hard_ham", 250)
    y_hard = np.zeros(hard_ham.shape[0])

    print("mails cargados")

    percentage = 0.5
    valPercentage = percentage + 0.25

    X = np.vstack((
                    getMatColPercent(spam, 0, percentage),
                    getMatColPercent(easy_ham, 0, percentage),
                    getMatColPercent(hard_ham, 0, percentage)
                ))
    y = np.hstack((
                    getMatPercent(y_spam, 0, percentage),
                    getMatPercent(y_easy, 0, percentage),
                    getMatPercent(y_hard, 0, percentage)
                ))

    Xval = np.vstack((
                    getMatColPercent(spam, percentage, valPercentage),
                    getMatColPercent(easy_ham, percentage, valPercentage),
```

```
                getMatColPercent(hard_ham, percentage, valPercentage)
            ))
    yval = np.hstack((
                getMatPercent(y_spam, percentage, valPercentage),
                getMatPercent(y_easy, percentage, valPercentage),
                getMatPercent(y_hard, percentage, valPercentage)
            ))
    Xtest = np.vstack((
                getMatColPercent(spam, valPercentage, 1),
                getMatColPercent(easy_ham, valPercentage, 1),
                getMatColPercent(hard_ham, valPercentage, 1)
            ))
    ytest = np.hstack((
                getMatPercent(y_spam, valPercentage, 1),
                getMatPercent(y_easy, valPercentage, 1),
                getMatPercent(y_hard, valPercentage, 1)
            ))

    svm, score, c, sigma = eleccionParams(X, y, Xval, yval)
    print("params elegidos")
    scoreTest = calculaScore(svm, Xtest, ytest)
    return scoreTest/Xtest.shape[0], c, sigma
```

Dividimos en entrenamiento (50), validación (25) y prueba (25) y calculamos los mejores valores para C y sigma (C=30, sigma=10).

En una iteración anterior hemos visto que si entramos con easy_ham le costará más evaluar hard_ham y viceversa, por ello lo hemos separado para que cada divisón tenga una parte de cada muestra. De esta manera conseguimos una precisión del 95.76%.