Practica 6 Aprendizaje Automático y Minería de Datos

Utilización de Support Vector Machines para casos básicos de clasificación y Filtrado de correos spam mediante el uso de SVMs. Por Mario Jimenez y Manuel Hernández

Importado de librerías

In [218]:

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from sklearn import svm
from mpl_toolkits.mplot3d import Axes3D
```

Conjunto de datos 1: Kernel Lineal

In [219]:

```
dataset1 = loadmat("ex6data1.mat")
X_1 = dataset1['X']
Y_1 = dataset1['y']

pos = np.array([X_1[i] for i in range(len(X_1))if Y_1[i] == 1])
neg = np.array([X_1[i] for i in range(len(X_1))if Y_1[i] == 0])
```

Inicialización del kernel lineal

In [220]:

```
kernel_lin = svm.SVC(C= 1, kernel = 'linear')
res = kernel_lin.fit(X_1, Y_1.flatten())
pred = kernel_lin.predict(np.array([[2, 4]]))

print(res, "\n")
print("Valor predicho para (2, 4): ", pred)
```

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Valor predicho para (2, 4):  [1]
```

Representación de datos

In [221]:

```
def dibujaDatos(pos, neg):
    plt.plot(pos[:, 0], pos [:, 1], 'gx', label = 'valores positivos')
    plt.plot(neg[:, 0], neg [:, 1], 'ro', label = 'valores negativos')
    plt.xlabel('Valores eje X')
    plt.ylabel('Valores eje Y')
    plt.legend()
    plt.grid(True)

def dibujaLimite(svm, pos, neg, minx, maxx, miny, maxy):
    figure = plt.figure()
    X = np.linspace(minx, maxx, 200)
    Y = np.linspace(miny, maxy, 200)
    zvals = np.zeros(shape = (len(X), len(Y)))

    for i in range(X.shape[0]):
        for j in range(Y.shape[0]):
            aux = np.array([[X[i],Y[j]]])
            zvals[i][j] = float(svm.predict(aux))

    zvals = zvals.T
    dibujaDatos(pos, neg)
    a, b = np.meshgrid (X, Y)
    contour = plt.contour (X, Y, zvals)
```

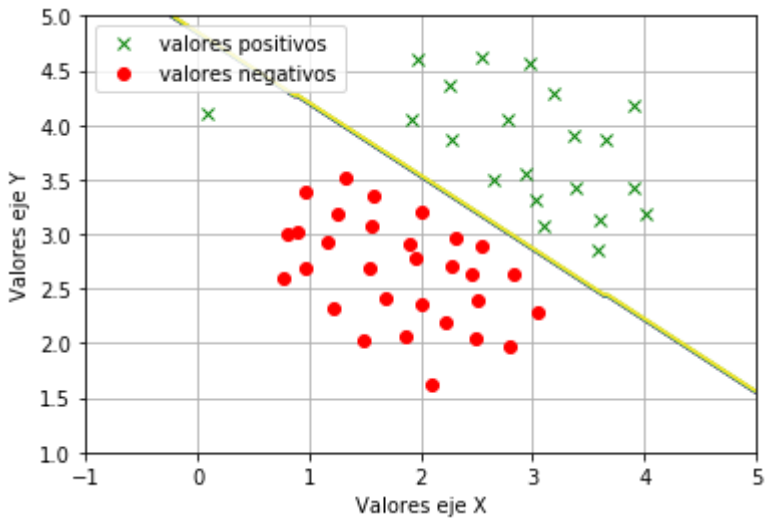
En los siguientes apartados podemos ver la diferencia, para el conjunto de datos proporcionado, de utilizar un coeficiente de penalización de C = 1 contra C = 100.

C = 1.0

In [222]:

```
plt.figure()
dibujaLimite(kernel_lin, pos, neg, -1,5,1,5)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



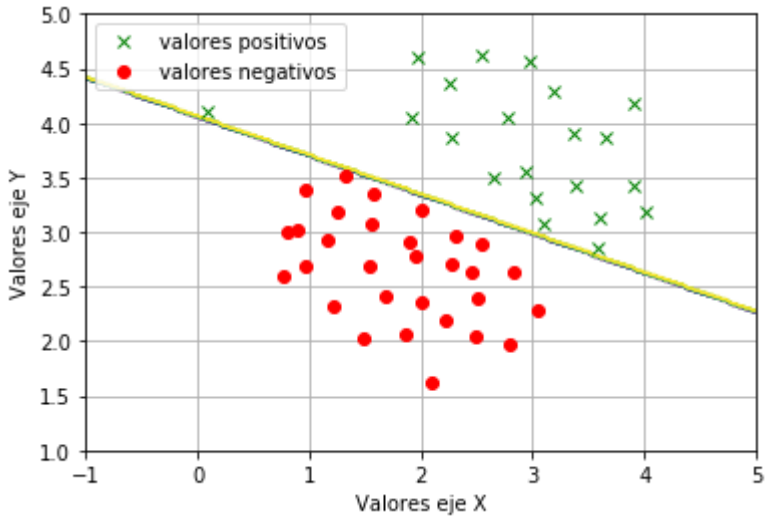
C = 100

In [223]:

```
kernel_lin = svm.SVC(C= 100, kernel = 'linear')
kernel_lin.fit(X_1, Y_1.flatten())

plt.figure()
dibujaLimite(kernel_lin, pos, neg, -1,5,1,5)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Conjunto de datos 2: Kernel Gaussiano

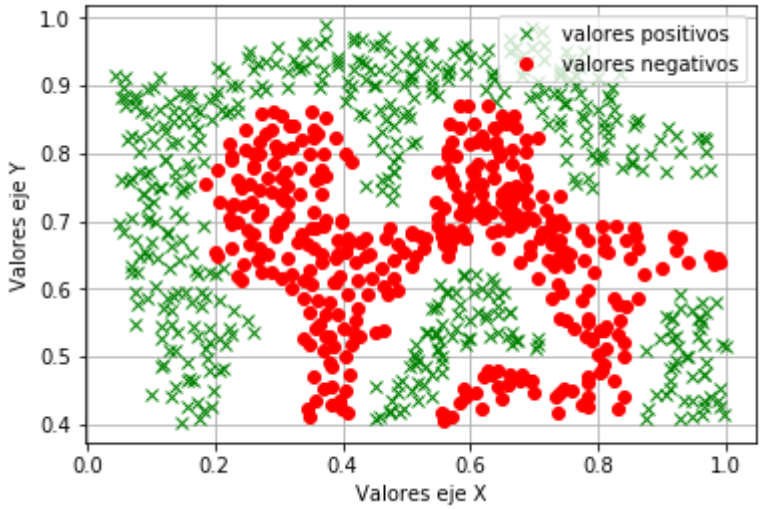
Obtención de datos

In [224]:

```
dataset2 = loadmat("ex6data2.mat")
X_2 = dataset2['X']
Y_2 = dataset2['y']

pos = np.array([X_2[i] for i in range(len(X_2))if Y_2[i] == 1])
neg = np.array([X_2[i] for i in range(len(X_2))if Y_2[i] == 0])


plt.figure()
dibujaDatos(pos, neg)
plt.show()
```



Entrenamiento del SVM

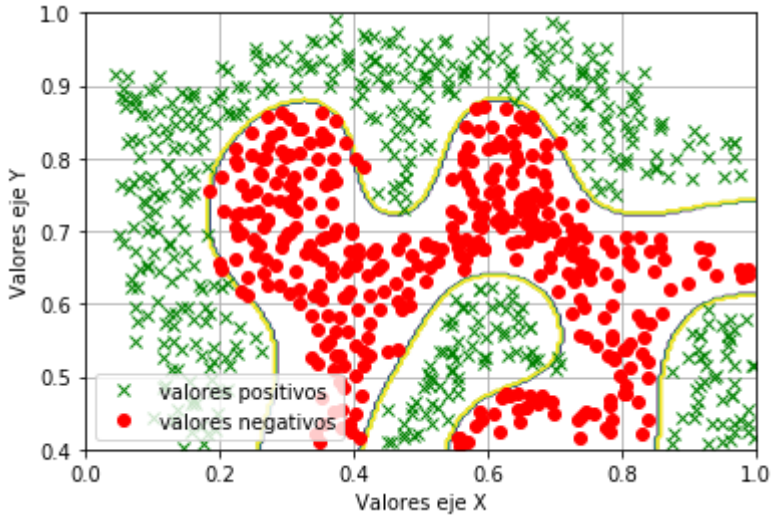
El SVM gaussiano recibe 2 parámetros que podemos ajustar:

- C: Penalización a elementos mal clasificados
- gamma: Influencia de cada ejemplo de entrenamiento en el resultado

In [225]:

```
sigma = 0.1
kernel_gaussiano = svm.SVC(C = 1, kernel = 'rbf', gamma = 1/(2*sigma **2))
kernel_gaussiano.fit(X_2, Y_2.flatten())

dibujaLimite(kernel_gaussiano, pos, neg, 0.0,1.0,0.4,1.0)
```



Conjunto de datos 3: Evaluación del modelo

En este apartado evaluaremos la precisión de un modelo mediante dos conjunto de datos distintos (entrenamiento y evaluación) y el ajuste de los parametros C y gamma.

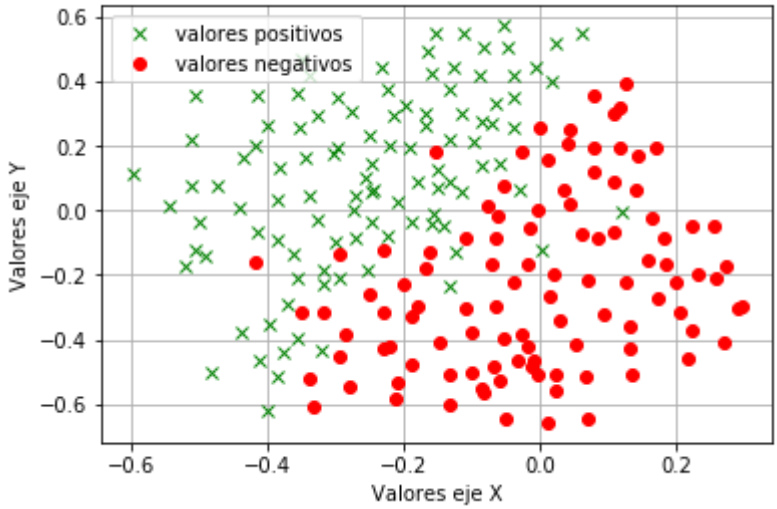
Obtencion de datos

In [238]:

```
dataset3 = loadmat("ex6data3.mat")
X_3 = dataset3['X']
Y_3 = dataset3['y']
X_Test = dataset3['Xval']
Y_Test = dataset3['yval']

pos_3 = np.array([X_3[i] for i in range(len(X_3))if Y_3[i] == 1])
neg_3= np.array([X_3[i] for i in range(len(X_3))if Y_3[i] == 0])

plt.figure()
dibujaDatos(pos_3, neg_3)
plt.show()
```



Trataremos ahora de obtener el modelo con menor porcentaje de fallos sobre el conjunto de datos. Para ello debemos comprobar, una vez entrenado, cuantos fallos y aciertos efectúa.

In [237]:

```
Cvals = np.array([0.01, 0.03, 0.1, 0.3, 1., 3., 10., 30.])
Sigmavals = np.copy(Cvals)


bestC = 0.01
bestSigma = 0.01
bestScore = -1

for C in Cvals:
    for sigma in Sigmavals:
        gamma = 1/(2*sigma **2)
        aux_kernel = svm.SVC(C = C, kernel = 'rbf', gamma = gamma)
        aux_kernel.fit(X_3, Y_3.flatten())
        score = aux_kernel.score(X_Test, Y_Test)
        if(score > bestScore):
            bestC = C
            bestSigma = sigma
            bestScore = score

print("Best score: ", bestScore)
print("Best sigma: ", bestSigma)
print("Best C:", bestC)
```

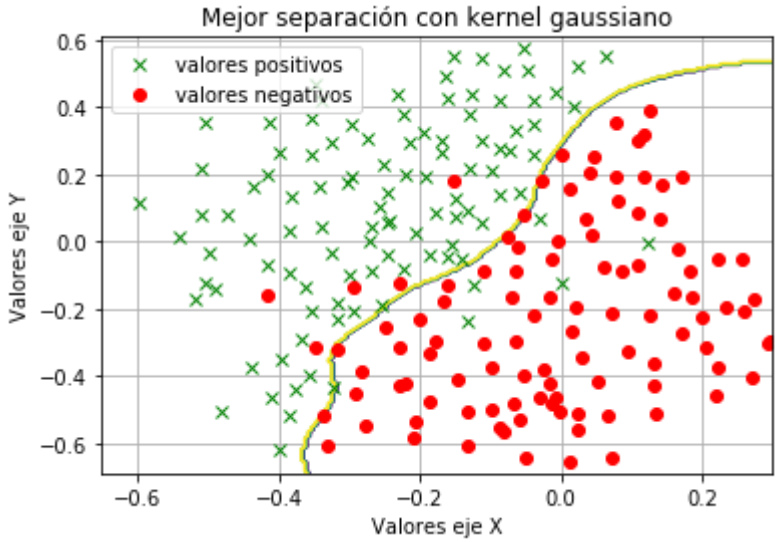
```
Best score:  0.965
Best sigma:  0.1
Best C: 1.0
```

In [242]:

```
gKernel = svm.SVC (C = bestC, kernel = 'rbf', gamma =  1/(2*bestSigma **2))
gKernel.fit(X_3, Y_3.flatten())

plt.figure()
dibujaLimite(gKernel, pos_3, neg_3, -0.65, 0.3, -0.69, 0.61)
plt.title("Mejor separación con kernel gaussiano")
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Parte 4: Filtrado de Spam de Correo

En esta sección entrenaremos un SVM para ayudarnos a filtrar correo spam de correo no spam.

Procesado del email

Cargamos un correo y lo procesamos como una lista de tokens, para poder entrenar el SVM.

In [258]:

```
import process_email as pmail
import get_vocab_dict as getvoc
import codecs

email_contents = codecs.open ("spam/0001.txt", 'r').read()
email = pmail.email2TokenList(email_contents)
print("Los correos se ven como listas de esta forma:\n", email)
```

```
Los correos se ven como listas de esta forma:
 ['save', 'up', 'to', 'number', 'on', 'life', 'insur', 'whi', 'spend', 'mo
re', 'than', 'you', 'have', 'to', 'life', 'quot', 'save', 'ensurin', 'g',
'your', 'famili', 's', 'financi', 'secur', 'is', 'veri', 'import', 'life',
'quot', 'save', 'ma', 'ke', 'buy', 'life', 'insur', 'simpl', 'and', 'affor
d', 'we', 'provid', 'free', 'access', 'to', 'the', 'veri', 'best', 'compan
i', 'and', 'the', 'lowest', 'rate', 'life', 'quot', 'save', 'is', 'fast',
'ea', 'y', 'and', 'save', 'you', 'money', 'let', 'us', 'help', 'you', 'ge
t', 'start', 'with', 'the', 'best', 'val', 'ue', 'in', 'the', 'countri',
'on', 'new', 'coverag', 'you', 'can', 'save', 'hundr', 'or', 'even', 'th
o', 'usand', 'of', 'dollar', 'by', 'request', 'a', 'free', 'quot', 'from',
'lifequot', 'save', 'our', 'servic', 'will', 'take', 'you', 'less', 'tha
n', 'number', 'minut', 'to', 'complet', 'shop', 'an', 'd', 'compar', 'sav
e', 'up', 'to', 'number', 'on', 'all', 'type', 'of', 'life', 'insur', 'cli
ck', 'here', 'for', 'your', 'free', 'quot', 'protect', 'your', 'famili',
'is', 'the', 'best', 'invest', 'you', 'll', 'eve', 'r', 'make', 'if', 'yo
u', 'are', 'in', 'receipt', 'of', 'thi', 'email', 'in', 'error', 'and', 'o
r', 'wish', 'to', 'be', 'remov', 'from', 'our', 'list', 'pleas', 'click',
'here', 'and', 'type', 'remov', 'if', 'you', 'resid', 'in', 'ani', 'stat
e', 'which', 'prohibit', 'e', 'mail', 'solicit', 'for', 'insuran', 'ce',
'pleas', 'disregard', 'thi', 'email']
```

In [292]:

```
import collections
vocabDict = collections.OrderedDict(getvoc.getVocabDict())
X_input = np.zeros(shape=(1000, len(vocabDict)))
Y = np.zeros(1000)

# Los 500 primeros valores son spam
for i in range(1,500):
    Y[i] = 1
    path = "spam/{:04d}.txt".format(i)
    email = pmail.email2TokenList(codecs.open (path, 'r', encoding = 'utf-8', errors =
'ignore').read())
    for j in range (len(email)):
        if(email [j] in vocabDict):
            X_input[i][list(vocabDict.keys()).index(email[j])] = 1

            # los 500 restantes no lo son
for i in range(1,500):
    Y[500+i] = 0
    path = "spam/{:04d}.txt".format(i)
    email = pmail.email2TokenList(codecs.open (path, 'r', encoding = 'utf-8', errors =
'ignore').read())
    for j in range (len(email)):
        if(email [j] in vocabDict):
            X_input[500+i][list(vocabDict.keys()).index(email[j])] = 1
```

In [293]:

```
spam_processor = svm.SVC(C = 1.0, kernel = 'linear')
spam_processor.fit(X_input, Y.ravel())
```

Out[293]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```