Práctica 6 (Support Vector Machines)

¿Cómo he planteado la práctica?

Primero necesitamos los datos, en este caso leemos los .MAT 'ex6data1', 'ex6data2' y 'ex6data3', y separamos los datos en X e Y y Xval e Yval.

En el paso siguiente comparamos los resultados de las distintas C.

Posteriormente creamos un Kernel Gaussiano y una función que nos permita escoger los mejores parámetros.

Finalmente mostramos el resultado con los mejores parámetros obtenidos.

Código comentado y gráficas

1. Importar las librerías

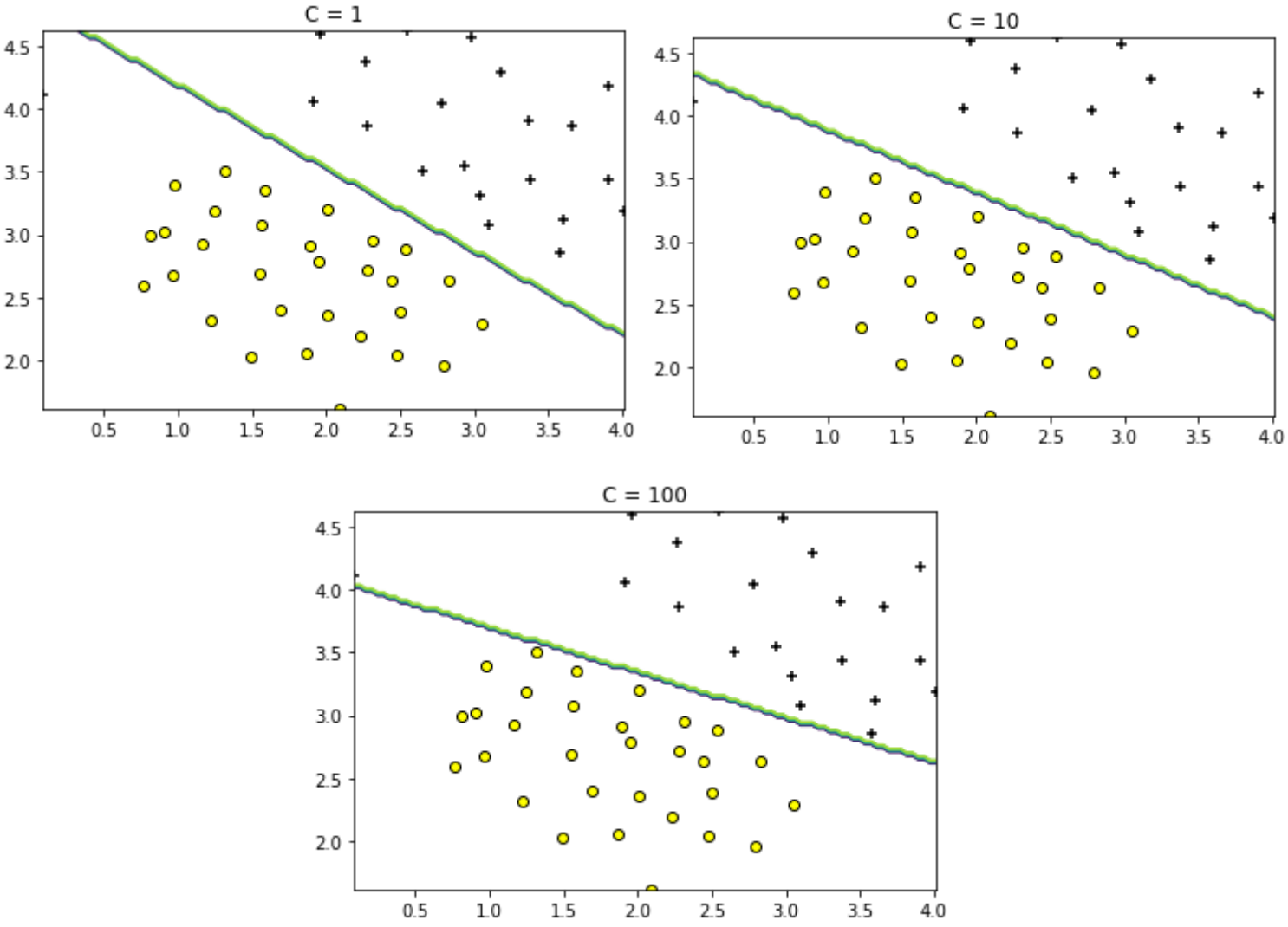
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import codecs


from scipy.io import loadmat
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from process_email import *
from get_vocab_dict import *

```

2. Obtener los datos del CSV:

```
data = loadmat('ex6data1.mat')
X1 = data['X']
Y1 = data['y'].ravel()

data = loadmat('ex6data2.mat')
X2 = data['X']
Y2 = data['y'].ravel()

data = loadmat('ex6data3.mat')
X3 = data['X']
Y3 = data['y'].ravel()
X3_val = data['Xval']
Y3_val = data['yval'].ravel()

```

3. Ahora comparamos los distintos resultados con las distintas C para el primer conjunto de datos.

```
svmLin1 = SVC(kernel = 'linear', C = 1)
svmLin1.fit(X1, Y1)
svmLin10 = SVC(kernel = 'linear', C = 10)
svmLin10.fit(X1, Y1)
svmLin100 = SVC(kernel = 'linear', C = 100)
svmLin100.fit(X1, Y1)

visualize_boundary(X1, Y1, svmLin1, 'C = 1')
visualize_boundary(X1, Y1, svmLin10, 'C = 10')
visualize_boundary(X1, Y1, svmLin100, 'C = 100')
```



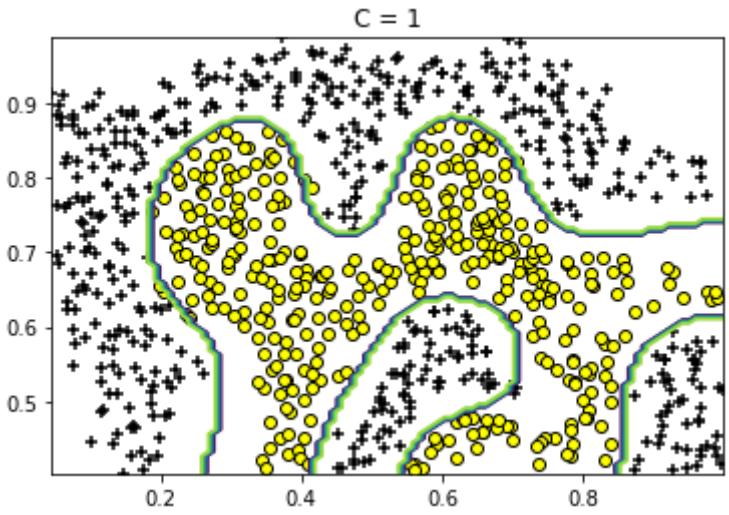
Como observación, creo que la que mejor se ajusta es C = 1, ya que es la que menos cerca está de todos los datos.

4. Ahora, para el segundo conjunto de datos, como no podemos hacer una función lineal, usaremos un kernel Gaussiano, en el enunciado especifica que C = 1 y $\sigma = 0'1$.

```
C = 1
sigma = .1
gamma = 1 / (2 * sigma**2)

svmGauss = SVC(kernel = 'rbf', C = C, gamma=gamma)
svmGauss.fit(X2, Y2)

visualize_boundary(X2, Y2, svmGauss, 'C = 1')
```

$$RBF \sim KERNEL\ GAUSSIANO$$

Y el resultado es:



5. Posteriormente implementamos una función que nos devuelve los mejores parámetros C y σ.

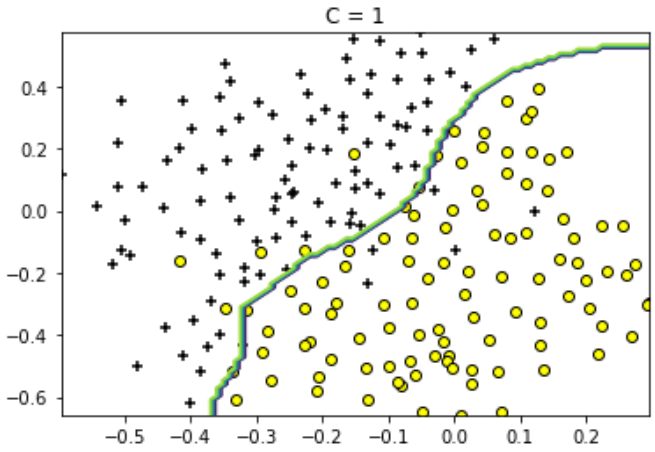
```
1  def parameter_election(X, Y, X_val, Y_val, kernel):
2      values = [ .01, .03, .1, .3, 1, 3, 10, 30 ]
3
4      cOpt = None
5      sOpt = None
6      accOpt = 0
7
8      for C in values:
9          for S in values:
10             gamma = 1 / (2 * S**2)
11
12             svm = SVC(kernel=kernel, C=C, gamma=gamma)
13             svm.fit(X, Y)
14             acc = accuracy_score(Y_val, svm.predict(X_val))
15
16             if(acc > accOpt):
17                 accOpt = acc
18                 svmOpt = svm
19                 cOpt = C
20                 sOpt = S
21
22     return accOpt, cOpt, sOpt
```

La función 'predict' (#14) predice los datos de un X, utilizando el modelo entrenado de la Support Vector Machine, por lo cual, para conocer la precisión de nuestra inteligencia artificial, debemos comparar los valores de Y_{VAL} con los valores predichos $\hat{Y}_{VAL}$ (X_{VAL})

6. Finalmente comprobamos el resultado y dibujamos la gráfica.

```
accOpt, cOpt, sOpt = parameter_election(X3, Y3, X3_val, Y3_val, 'rbf')

print("La C óptima es: {} y la sigma óptima es: {}\n\tcon un {}% de precisión"
      .format(cOpt, sOpt, np.round(accOpt * 100), 2))


svmOpt = SVC(kernel = 'rbf', C = cOpt, gamma=1 / (2 * sOpt**2))
svmOpt.fit(X3, Y3)
visualize_boundary(X3, Y3, svmOpt, 'C = ' + str(cOpt))
```

```
La C óptima es: 1 y la sigma óptima es: 0.1
        con un 96.0% de precisión
```



Práctica 6 (Detector de Spam)

¿Cómo he planteado la práctica?

Primero necesitamos los datos, en este caso leemos los directorios 'spam', 'easy_ham' y 'hard_ham'. Para ello nos apoyaremos en una función

En el paso siguiente, juntamos las X de los anteriores archivos y las dividimos en dos subconjuntos, X y X_{VAL}.

Posteriormente creamos las Y, con el mismo tamaño que X y X_{VAL}.

Finalmente entrenamos la Inteligencia Artificial y obtenemos su porcentaje de acierto.

Código comentado y gráficas

1. Importar las librerías

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import os
5  import codecs
6
7
8  from scipy.io import loadmat
9  from sklearn.svm import SVC
10 from sklearn.metrics import accuracy_score
11 from process_email import *
12 from get_vocab_dict import *
13
```

2. Creamos una función que nos devuelva la X a partir de varios archivos, cada uno tiene varios emails:

```
1  def read_dir(name='spam'):
2      dic = getVocabDict()
3      X = []
4
5      for file in os.listdir(name):
6          fileName = name + '/' + file
7          email = email2TokenList(
8                  codecs.open(fileName,
9                              'r',
10                             encoding='utf-8',
11                             errors='ignore'
12                            ).read()
13         )
14
15         temp = np.zeros(len(dic.keys()))
16
17         for v in dic.keys():
18             if(v in email):
19                 temp[dic[v] - 1] = 1
20
21
22         X.append(temp)
23
24     return np.array(X)
```

Primero obtenemos el diccionario de vocabulario (#2) y leemos todos los archivos(#5). Después inicializamos un array de ceros(#15), en el que, en cada entrada del diccionario(#17), comprobamos si existe en el email leído(#18), si existe, lo ponemos a 1(#19), si no, nada. Finalmente devolvemos todas las X en un array(#24).

3. Leemos los archivos 'spam', 'easy_ham' y 'hard_ham'.

```
XSpam = read_dir('spam')
XEasyHam = read_dir('easy_ham')
XHardHam = read_dir('hard_ham')
```

4. Ahora establecemos un tamaño para el Train Set, en este caso será de un 80%, frente a un 20% de Cross Validation.
   Y separamos el Train Set (#9) del CV Set (#13)

```
size = .8

# Join X and separate data

XSpamSize = int(XSpam.shape[0] * size)
XEasyHamSize = int(XEasyHam.shape[0] * size)
XHardHamSize = int(XHardHam.shape[0] * size)

X = np.concatenate((XSpam[:XSpamSize],
                    XEasyHam[:XEasyHamSize],
                    XHardHam[:XHardHamSize]))

X_val = np.concatenate((XSpam[XSpamSize:],
                    XEasyHam[XEasyHamSize:],
                    XHardHam[XHardHamSize:]))
```

5. Posteriormente reutilizamos la función 'parameter_election' de la anterior parte de la práctica, esta función nos devolvía la $precisión_{OPT}$, la C_{OPT} y la σ_{OPT}.

```
acc, C, sigma = parameter_election(X, Y, X_val, Y_val, 'rbf')
```

NOTA: Lógicamente usamos el Kernel Gaussiano, ya que la distribución de datos no puede ser lineal.

6. Finalmente mostramos la precisión obtenida:

```
print("La mejor precisión obtenida es de un {}%\ncon C = {} y sigma = {}"
      .format(np.round(acc * 100, 2), C, sigma
))
```

```
La mejor precisión obtenida es de un 97.43%
con C = 3 y sigma = 10
```