Práctica 6: Support Vector Machines

Álvar Domingo Fernández y Pablo Jurado López

Preparación inicial

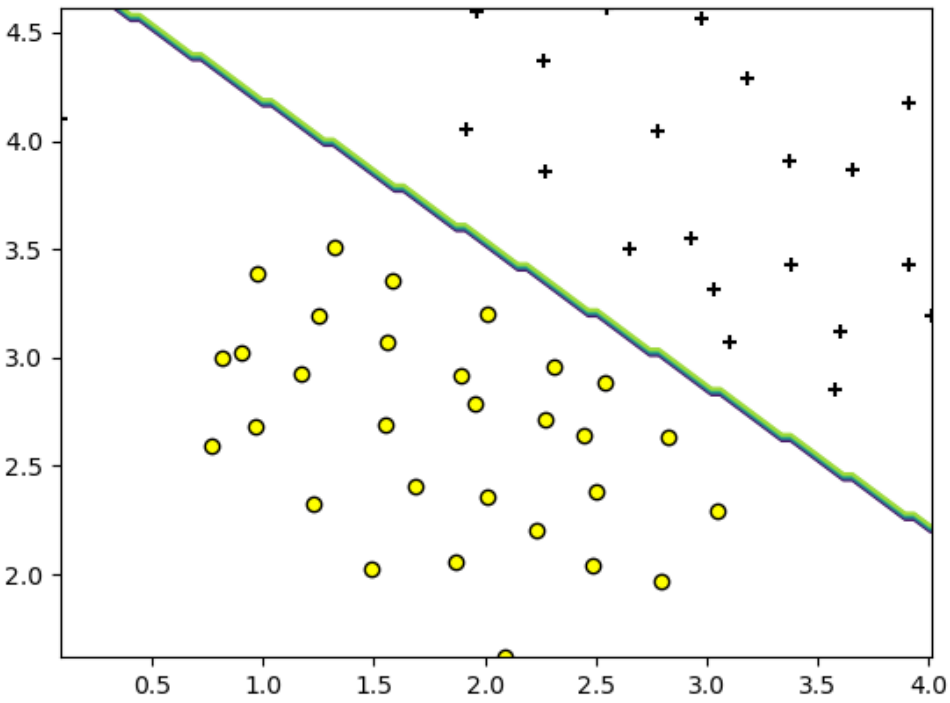
A continuación se importan todas las librerías y funciones externas que serán utilizadas en esta práctica.

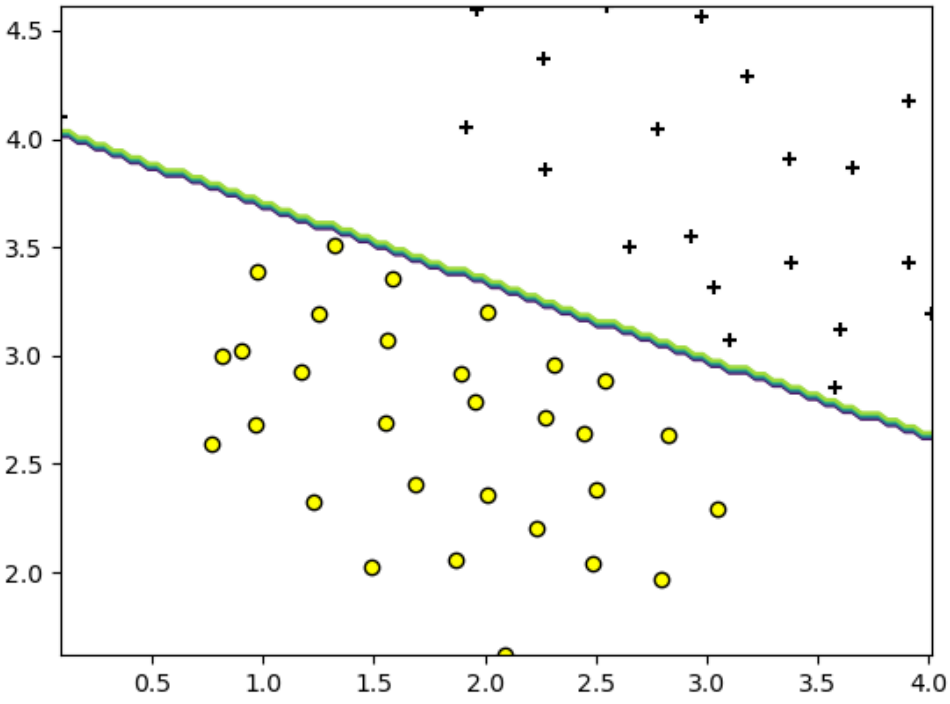
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from scipy.optimize import minimize
from sklearn.svm import SVC
from process_email import email2TokenList
import codecs
from get_vocab_dict import getVocabDict
import glob
import sklearn.model_selection as ms
from sklearn.metrics import confusion_matrix
import time
```

1.1- Kernel lineal

Hemos clasificado los datos contenidos en ex6data1.mat con un SVM con un parámetro de regularización C=1 y una función de kernel lineal, y lo hemos renderizado en una gráfica. Posteriormente hemos repetido el mismo proceso, esta vez con un parámetro de regularización C = 100.

[ ]:
```
def apartado1_1():
    data = loadmat('ex6data1.mat')
    X, y = data['X'], data['y'].ravel()
    svm = SVC(kernel= 'linear', C = 1)
    visualize_boundary(X, y, svm, "apartado1_1_C1")
    svm = SVC(kernel= 'linear', C = 100)
    visualize_boundary(X, y, svm, "apartado1_1_C100")
```

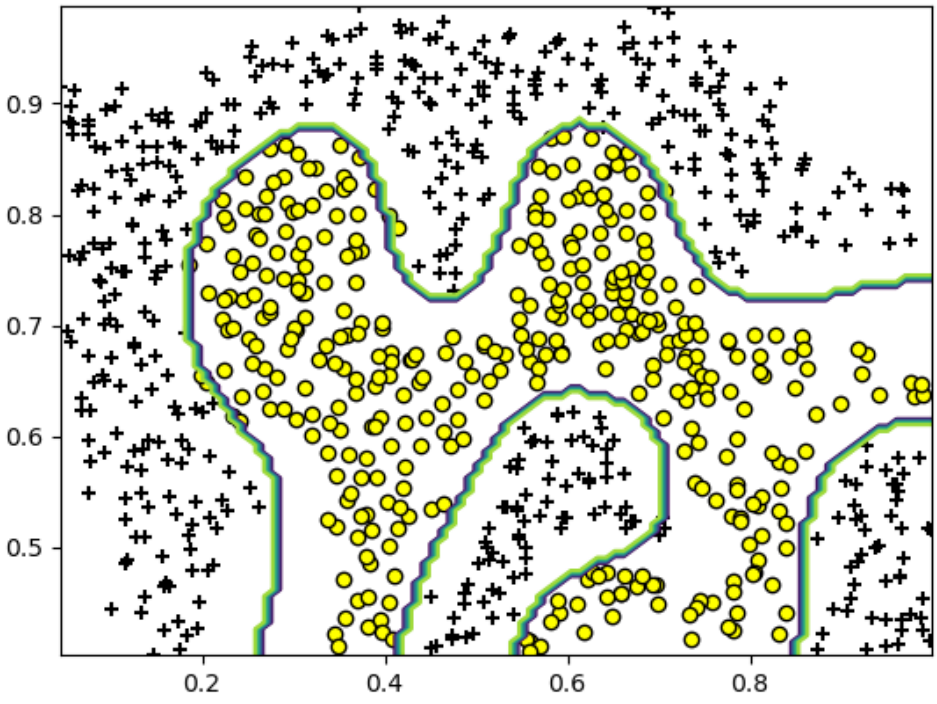

4.5
4.0
3.5
3.0
2.5
2.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



1.2- Kernel gaussiano

Hemos repetido el proceso del apartado anterior, pero esta vez hemos utilizado una función de kernel gaussiana

[ ]:
```
def apartado1_2():
    data = loadmat('ex6data2.mat')
    X, y = data['X'], data['y'].ravel()
    C = 1
    sigma = 0.1
    svm = SVC(kernel= 'rbf', C = C, gamma=1 / (2 * sigma**2))
    visualize_boundary(X, y, svm, "apartado1_2")
```



1.3- Elección de los parámetros C y σ

Hemos utilizado la función svm.score y probado con los siguientes valores para C y σ: [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

[ ]:
```
def apartado1_3():
    data = loadmat('ex6data3.mat')
    X = data['X']
    y = data['y'].ravel()
    Xval = data['Xval']
    yval = data['yval'].ravel()
    values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    n = len(values)
    scores = np.zeros((n, n))

    for i in range(n):
        C = values[i]
        for j in range(n):
            sigma = values[j]
            svm = SVC(kernel='rbf', C = C, gamma= 1 / (2 * sigma **2))
            svm.fit(X, y.ravel())
            scores[i, j] = svm.score(Xval, yval)
```

```
    print("Error mínimo: {}".format(1 - scores.max()))
    C_opt = values[scores.argmax()//n]
    sigma_opt = values[scores.argmax()%n]
    print("C óptimo: {}, sigma óptimo: {}".format(C_opt, sigma_opt))

    svm = SVC(kernel= 'rbf', C = C_opt, gamma= 1 / (2 * sigma_opt ** 2))
    visualize_boundary(X, y, svm, "apartado1_3")
```

El resultado obtenido es el siguiente:
Error mínimo: 0.03500000000000003
C óptimo: 1, σ óptimo: 0.1

2 - Detección de spam

Para leer los datos hemos utilizado una función llamada *read_directory*, escrita a continuación:

[ ]:
```
def read_directory(name, vocab):
    files = glob.glob(name)
    X = np.zeros((len(files), len(vocab)))
    y = np.ones(len(files))
    i = 0
    for f in files:
        email_contents = codecs.open(f, 'r', encoding='utf-8', errors='ignore').
↪read()
        tokens = email2TokenList(email_contents)
        words = filter(None,[vocab.get(x) for x in tokens])
        for w in words:
            X[i, w-1] = 1
        i +=1
    return X, y
```

Lo que hace es abrir todos los archivos de cierto directorio, cuenta las veces que sale cada palabra del vocabulario y devuelve una matriz "X" con estos datos y una matriz "y" de unos.

[ ]:
```
def palabras_comunes(X, fracción):
    num_palabras = int(np.round(len(X[0])*fracción))
    frecuencia = np.sort(X.sum(0))[-num_palabras]
    atributos = np.where(X.sum(0) >= frecuencia)[0]
    return X[:, atributos]
```

La función *palabras_comunes* saca una submatriz de la original en la que solo están las palabras más comunes según la fracción del total que se indique.

[ ]:
```
def score_emails(X, y):
    X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size=0.2,␣
↪random_state=1)
    X_train, X_val, y_train, y_val = ms.train_test_split(X_train, y_train,␣
↪test_size=0.25, random_state=1)
```

```
    values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100, 300]
    n = len(values)
    scores = np.zeros((n, n))

    startTime = time.process_time()

    for i in range(n):
        C = values[i]
        for j in range(n):
            sigma = values[j]
            svm = SVC(kernel='rbf', C = C, gamma= 1 / (2 * sigma **2))
            svm.fit(X_train, y_train)
            scores[i, j] = svm.score(X_val, y_val)

    print("Error mínimo: {}".format(1 - scores.max()))
    C_opt = values[scores.argmax()//n]
    sigma_opt = values[scores.argmax()%n]
    print("C óptimo: {}, sigma óptimo: {}".format(C_opt, sigma_opt))

    svm = SVC(kernel='rbf', C= C_opt, gamma=1 / (2 * sigma_opt)**2)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    endTime = time.process_time()

    score = svm.score(X_test, y_test)
    totalTime = endTime - startTime

    print('Precisión: {:.3f}%'.format(score*100))
    print('Tiempo de ejecución: {}'.format(totalTime))
    print('Matriz de confusión: ')
    print(confusion_matrix(y_test, y_pred))
```

score_emails utiliza SVC para clasificar los emails e intenta buscar el C y σ óptimos de entre estos valores: [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100, 300]. También se guarda el tiempo guardado, la precisión y la matriz de confusión. A continuación veremos la diferencia de resultados entre hacer reducción de atributos o no.

[ ]:
```
def apartado2():
    vocab = getVocabDict()
    Xspam, yspam = read_directory('spam/*.txt', vocab)

    Xeasy_ham, yeasy_ham = read_directory('easy_ham/*.txt', vocab)

    Xhard_ham, yhard_ham = read_directory('hard_ham/*.txt', vocab)

    X = np.vstack((Xspam, Xeasy_ham, Xhard_ham))
```

```
    yspam = [1]*len(Xspam)
    yeasy_ham = [0]*len(Xeasy_ham)
    yhard_ham = [0]*len(Xhard_ham)
    y = np.r_[yspam, yeasy_ham, yhard_ham]

    # con reducción de atributos
    X_freq = palabras_comunes(X, 0.1)

    score_emails(X_freq, y)

    # sin reducción de atributos

    score_emails(X, y)
```

Recolectamos los datos de los tres directorios y primero utilizamos *score_emails* con reducción de atributos, y después sin ella. Los resultados obtenidos son los siguientes:

Con reducción de atributos:

Error mínimo: 0.025757575757575757
C óptimo: 1, σ óptimo: 3
Precisión: 97.428%
Tiempo de ejecución: 42.3125
Matriz de confusión:
[[550 5]
[12 94]]

Sin reducción de atributos:

Error mínimo: 0.015151515151515138
C óptimo: 10, σ óptimo: 10
Precisión: 98.185%
Tiempo de ejecución: 379.59375
Matriz de confusión:
[[551 4]
[8 98]]