Práctica 6

Alejandro Aizel Boto
Miguel Robledo Blanco

Índice

1. Descripción General

En esta práctica, se nos pedía aplicar los conceptos de SVM sobre 3 conjuntos de datos distintos. Un primer conjunto de datos linealmente separable (kernel lineal), un segundo conjunto de datos no linealmente separable (kernel gaussiano) y un tercer conjunto de datos tampoco linealmente separable pero que requería iterar para encontrar la configuración óptima.

El segundo apartado de la práctica consiste en un detector de SPAM. Utilizando lo que hemos aprendido en los apartados anteriores tenemos que crear distintos clasificadores utilizando diferentes sets de entrenamiento.

2. Explicación de la Solución

Comenzamos con el método `main()` que tiene dos llamadas a otros dos métodos `support_vector_machines()` y `spam_detection()` que hacen referencia al primer y segundo punto de la práctica de forma respectiva. Comenzamos con la llamada a la función `support_vector_machines()` que a su vez llama a otras tres funciones correspondientes a los subapartados del punto primero.

```
def main():
    support_vector_machines()
    spam_detection()


def support_vector_machines():
    lineal_kernel()
    gaussian_kernel()
    parameters_choice()
```

Empezamos con la funcón `lineal_kernel()` en la que leemos los datos del primer conjunto y los extraemos del diccionario. Instanciamos la clase `SVC` con un kernel lineal con `C=1`, hacemos `fit()` para que se ajuste a nuestros datos de entrenamiento y visualizamos la frontera. Después volvemos a hacer lo mismo pero con `C=100`.

```
def lineal_kernel():
    data = loadmat('Práctica 6/Recursos/ex6data1.mat')

    X, Y = data['X'], data['y'].ravel()

    svm = sk.SVC(kernel='linear', C=1)
    svm.fit(X,Y)

    show_contour(X, Y, svm,'lineal_kernel_c_1')

    svm = sk.SVC(kernel='linear', C=100)
    svm.fit(X, Y)

    show_contour(X, Y, svm, 'lineal_kernel_c_100')
```

Por último llamamos a la función `visualize_boundary()` para mostrar la gráfica.

```
def visualize_boundary(X, y, svm, name ='visualize boundary'):
    x1 = np.linspace(X[:, 0].min()-0.1, X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min()-0.1, 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('Práctica 6/Recursos/{}.png'.format(name), dpi=200)
    plt.close()
```

Esta función es la proporcionada por el profesor para mostrar la gráfica y no merece mayor explicación ya que se explicó una función similar en prácticas anteriores.

A continuación llamamos a la segunda función `gaussian_kernel()` que tiene la siguiente forma.

```
def gaussian_kernel():
    data = loadmat('Práctica 6/Recursos/ex6data2.mat')

    X, Y = data['X'], data['y'].ravel()

    sigma = 0.1

    svm = sk.SVC(kernel='rbf', C=1, gamma=1 / (2 * sigma ** 2))
    svm.fit(X, Y)

    visualize_boundary(X, Y, svm, 'gaussian_kernel_c_1_s_0.1')
```

El objetivo del kernel gaussiano es separar un conjunto de datos que no es linealmente separable, para ello se hace un cambio de dimensión en el cual el conjunto de datos si es linealmente separable.

El cambio de dimensión se realiza calculando la distancia desde un punto (perteneciente al conjunto de datos de entrenamiento) a todos los demás obteniendo un vector de distancias el cual son las coordenadas del nuevo punto.

Por último, para el primer apartado, llamamos a la función `parameters_choice()` que tiene la siguiente forma.

```
def parameters_choice():
    data = loadmat('Práctica 6/Recursos/ex6data3.mat')

    X, Y = data['X'], data['y'].ravel()
    X_val, Y_val = data['Xval'], data['yval'].ravel()

    values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
```

```
    C_aux, sigma_aux = np.meshgrid(values, values)

    combinations = np.vstack((C_aux.ravel(), sigma_aux.ravel())).T
    errors = []

    for v in combinations:
        c = sk.SVC(kernel='rbf', C=v[0], gamma=1 / (2 * v[1]**2))
        c.fit(X, Y)

        errors.append(c.score(X_val, Y_val))

    best_value = combinations[np.argmax(errors)]

    best = sk.SVC(kernel='rbf', C=best_value[0], gamma=1 / (2 *
         best_value[1] ** 2))
    best.fit(X, Y)

    show_contour(X, Y, best, 'parameter_selection')
```

La función comienza haciendo un `meshgrid` con el vector de values sobre sí mismo y haciendo un `vstack` del resultado. De esta forma obtenemos una matriz de 64 filas por 2 columnas, obteniendo todas las combinaciones posibles de `C` y `sigma`. Aquí usamos la función `score()` sobre el conjunto de validación cruzada. Esta función calcula el acierto medio del `SVC`, de esta forma, para cada uno de las 64 posibles combinaciones, instanciamos un `SVC` nuevo, y calculamos el acierto medio para `C` y `sigma`.

Como queremos quedarnos con la combinación que acierte mejor, hacemos `np.argmax()` para obtener el índice en el que se encuentra el valor máximo. Una vez tenemos el valor de `C` y `sigma` óptimo, volvemos a hacer una predicción y mostramos el resultado.

Por último tenemos el apartado 2, el detector de SPAM. Para esto hemos creado la función `spam_detection()` que tiene la siguiente forma.

```
def spam_detection():
    training_set(500, 1551, 0, "first")
    training_set(400, 1300, 150, "second")
    training_set(350, 1200, 250, "third")
```

Esta función lo que hace es llamar a otra llamada `training_set()` que hace el entrenamiento y validación con distintos parámetros que se pasan. El primero indica el número de emails de spam que se van a usar para el entrenamiento, el segundo el número de emails de `easy_ham` que se van a usar para el entrenamiento y el tercero el de `hard_ham` que se van a usar para el entrenamiento. El último parámetro es un nombre para poder identificar cada set de entrenamiento. Todos los emails que no se usen para el entrenamiento se usarán para el test. La función `training_set()` tiene esta forma.

```
def training_set(num_spam_train, num_easy_ham_train, num_hard_ham_train,
name):
    vocab_dict = gvd.getVocabDict()

    X_train_spam = make_data("spam", 0, num_spam_train, vocab_dict)
    Y_train_spam = np.ones(num_spam_train)

    X_train_easy_ham = make_data("easy_ham", 0, num_easy_ham_train,
        vocab_dict)
    Y_train_easy_ham = np.zeros(num_easy_ham_train)

    X_train_hard_ham = make_data("hard_ham", 0, num_hard_ham_train,
        vocab_dict)
    Y_train_hard_ham = np.zeros(num_hard_ham_train)

    X_train = np.concatenate((X_train_spam, X_train_easy_ham,
        X_train_hard_ham), axis=0)
    Y_train = np.concatenate([Y_train_spam, Y_train_easy_ham,
        Y_train_hard_ham])

    X_val_spam = make_data("spam", num_spam_train, 500, vocab_dict)
    Y_val_spam = np.ones(500 - num_spam_train)

    X_val_easy_ham = make_data("easy_ham", num_easy_ham_train, 1551,
        vocab_dict)
    Y_val_easy_ham = np.zeros(1551 - num_easy_ham_train)

    X_val_hard_ham = make_data("hard_ham", num_hard_ham_train, 250,
        vocab_dict)
    Y_val_hard_ham = np.zeros(250 - num_hard_ham_train)

    X_val = np.concatenate((X_val_spam, X_val_easy_ham, X_val_hard_ham),
        axis=0)
    Y_val = np.concatenate([Y_val_spam, Y_val_easy_ham, Y_val_hard_ham])

    svm = sk.SVC(kernel='linear', C=1)
    svm.fit(X_train, Y_train)

    accuracy = svm.score(X_val, Y_val)

    print("\{} new training set:".format(name))
    print("   Correctly classifed for C=1: {} %".format(round(accuracy *
        100, 2)))

    svm = sk.SVC(kernel='linear', C=100)
    svm.fit(X_train, Y_train)

    accuracy = svm.score(X_val, Y_val)

    print("   Correctly classifed for C=100: {} %".format(round(accuracy
        * 100, 2)))
```

Primero usamos la función `getVocabDict()` para conseguir el diccionario de palabras. Seguidamente usamos la función `make_data()` para generar cada matriz de datos tanto para el entrenamiento como para la validación, que luego juntamos con `concatenate()`. Para las `Y` creamos un array de 0 o 1 según sea ham o spam y lo juntamos

en orden para que coincida con su respectiva `X`. Una vez tenemos los datos vamos a hacer dos entrenamientos uno para `C=1` y otro para `C=100`. Inicializamos el `SVC` con kernel lineal y hacemos `fit()`. Por último sacamos la exactitud de la predicción con los datos de validación y lo imprimimos en forma de porcentaje. La función `make_data()` tiene la siguiente forma.

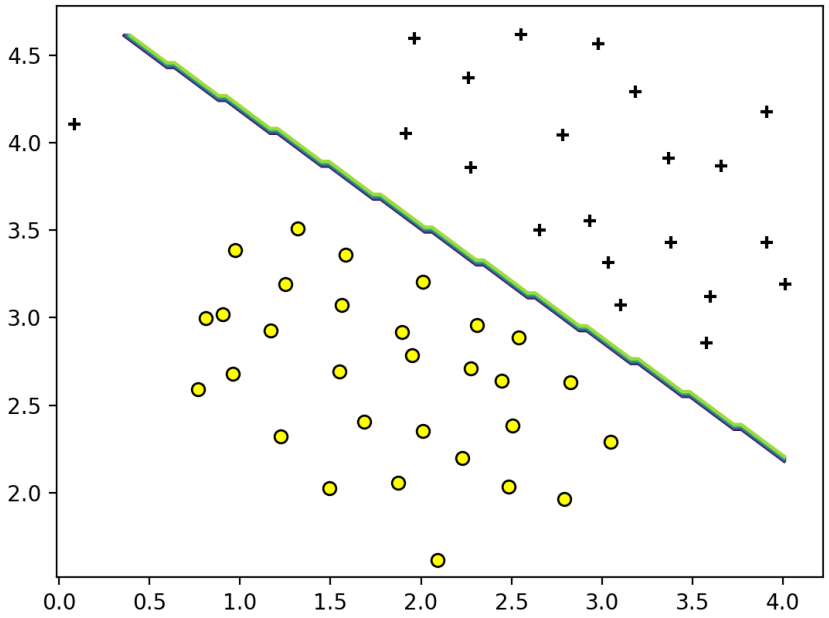
```
def make_data(directory, num_emails_begin, num_emails_end, vocab_dict):
    email_features = np.zeros((num_emails_end - num_emails_begin,
         len(vocab_dict)))

    for i in range(num_emails_begin, num_emails_end):
        email_contents = cod.open('Práctica 6/Recursos/{}/
             {}.txt'.format(directory, f'{(i + 1):04}'), 'r',
                  encoding='utf-8', errors='ignore').read()
        email = pe.email2TokenList(email_contents)

        for w in email:
            if w in vocab_dict:
                email_features[i - num_emails_begin, vocab_dict[w] - 1] = 1

    return email_features
```

Lo que hacemos en esta función es primeramente crear una matriz de 0 con tamaño la sección de emails que vamos a tratar. Ahora lo que hacemos es ir uno a uno con un for y tras leer el email con `open()` lo transformamos con `email2TokenList()` a un array de palabras. Finalmente vamos recorriendo cada palabra de ese array y mirando si está en el diccionario de palabras anteriormente mencionado. En caso de que esté se coloca un 1 en esa posición de la palabra en su correspondiente email.

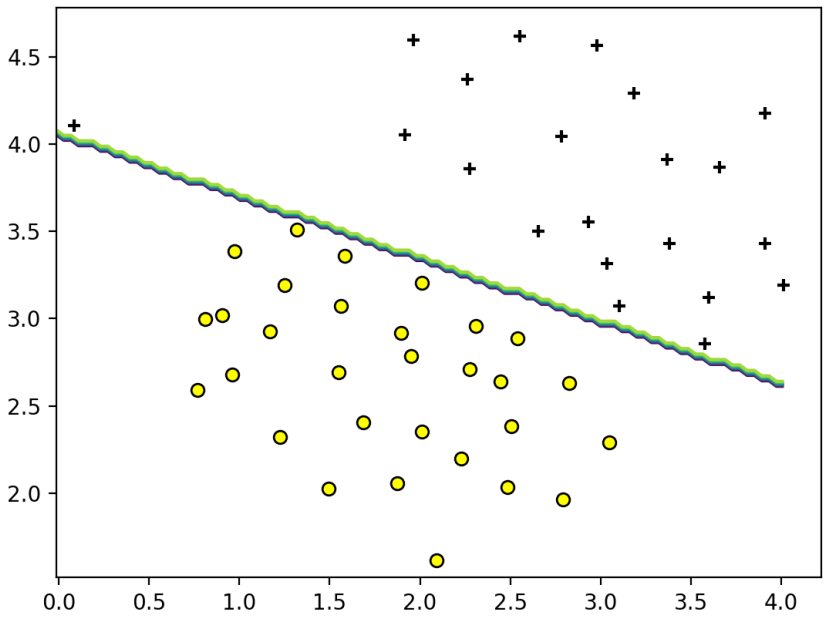
3. Resultados

En este apartado vamos a ir comentando los distintos resultados que hemos obtenido.

En el primer apartado obtenemos dos gráficas diferentes, la primera con un valor de `C=1` y la segunda con un valor de `C=100`. Se muestran a continuación.

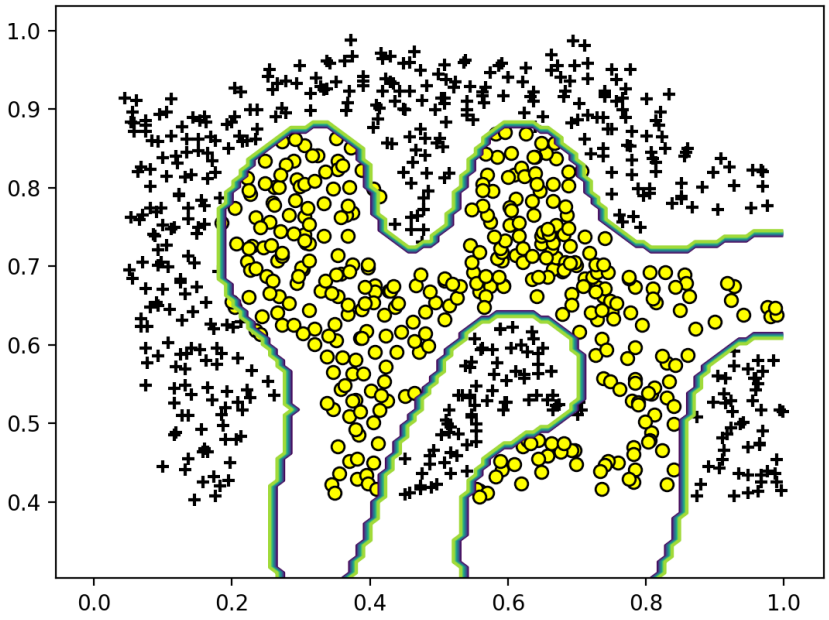


C=1

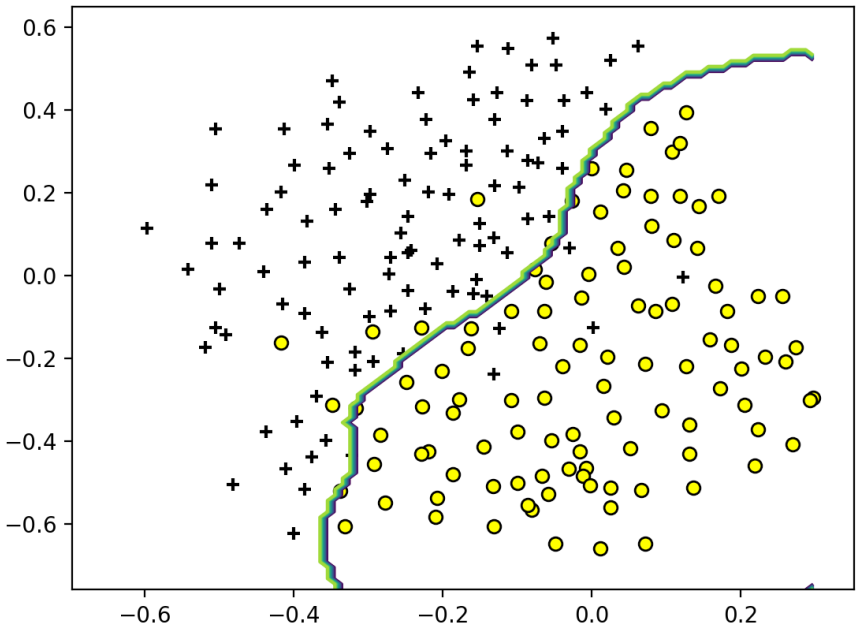


C=100

En el segundo apartado del punto 1 obtenemos la gráfica siguiente.



En el tercer apartado del punto 1 obtenemos la siguiente gráfica



Para la parte de emails como comentamos en la sección de explicación del código tenemos 3 sets de entrenamiento. Para cada uno tenemos 2 entrenamientos con distinto valor de C, a continuación se muestran los resultados de llamar a cada función.

```
> training_set(500, 1551, 0, "first")
```

```
   first training set:
      Correctly classified for C=1: 48.8 %
      Correctly classified for C=100: 55.2 %
```

```
> training_set(400, 1300, 150, "second")
```

```
   second training set:
      Correctly classified for C=1: 90.47 %
      Correctly classified for C=100: 90.69 %
```

```
> training_set(350, 1200, 250, "third")
```

```
   third training set:
      Correctly classified for C=1: 96.81 %
      Correctly classified for C=100: 95.81 %
```

4. Conclusión

Esta práctica nos ha servido para familiarizarnos con el uso del clasificador SVM que incorpora sckit-learn. Además hemos usado esto para luego aplicarlo en la segunda parte de la práctica en la que experimentamos con la detección de correo spam.

En el siguiente apartado se muestra el código completo por si se quiere ver seguido o hacer alguna prueba compilada.

5. Código

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import Recursos.process_email as pe
import Recursos.get_vocab_dict as gvd
from scipy.io import loadmat
from scipy.optimize import minimize
from sklearn import svm as sk
import codecs as cod

def visualize_boundary(X, y, svm, name ='visualize boundary'):
    x1 = np.linspace(X[:, 0].min()-0.1, X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min()-0.1, 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('Práctica 6/Recursos/{}.png'.format(name), dpi=200)
    plt.close()

# 1.1. Kernel lineal
def lineal_kernel():
    data = loadmat('Práctica 6/Recursos/ex6data1.mat')

    X, Y = data['X'], data['y'].ravel()

    svm = sk.SVC(kernel='linear', C=1)
    svm.fit(X,Y)

    visualize_boundary(X, Y, svm,'lineal_kernel_c_1')

    svm = sk.SVC(kernel='linear', C=100)
    svm.fit(X, Y)

    visualize_boundary(X, Y, svm, 'lineal_kernel_c_100')

# 1.2. Kernel gaussiano
def gaussian_kernel():
    data = loadmat('Práctica 6/Recursos/ex6data2.mat')

    X, Y = data['X'], data['y'].ravel()

    sigma = 0.1

    svm = sk.SVC(kernel='rbf', C=1, gamma=1 / (2 * sigma ** 2))
    svm.fit(X, Y)

    visualize_boundary(X, Y, svm, 'gaussian_kernel_c_1_s_0.1')
```

```
# 1.3. Elección de los parámetros C y sigma
def parameter_selection():
    data = loadmat('Práctica 6/Recursos/ex6data3.mat')

    X, Y = data['X'], data['y'].ravel()
    X_val, Y_val = data['Xval'], data['yval'].ravel()

    values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

    C_aux, sigma_aux = np.meshgrid(values, values)

    combinations = np.vstack((C_aux.ravel(), sigma_aux.ravel())).T
    errors = []

    for v in combinations:
        c = sk.SVC(kernel='rbf', C=v[0], gamma=1 / (2 * v[1]**2))
        c.fit(X, Y)

        errors.append(c.score(X_val, Y_val))

    best_value = combinations[np.argmax(errors)]

    best = sk.SVC(kernel='rbf', C=best_value[0], gamma=1 / (2 * best_value[1]
** 2))
    best.fit(X, Y)

    visualize_boundary(X, Y, best, 'parameter_selection')


def make_data(directory, num_emails_begin, num_emails_end, vocab_dict):
    email_features = np.zeros((num_emails_end - num_emails_begin,
len(vocab_dict)))

    for i in range(num_emails_begin, num_emails_end):
        email_contents = cod.open('Práctica 6/Recursos/{}/
{}.txt'.format(directory, f'{(i + 1):04}'), 'r',
            encoding='utf-8', errors='ignore').read()
        email = pe.email2TokenList(email_contents)

        for w in email:
            if w in vocab_dict:
                email_features[i - num_emails_begin, vocab_dict[w] - 1] = 1

    return email_features

# 2. Detección de spam
def training_set(num_spam_train, num_easy_ham_train, num_hard_ham_train,
name):
    vocab_dict = gvd.getVocabDict()

    X_train_spam = make_data("spam", 0, num_spam_train, vocab_dict)
    Y_train_spam = np.ones(num_spam_train)

    X_train_easy_ham = make_data("easy_ham", 0, num_easy_ham_train,
vocab_dict)
    Y_train_easy_ham = np.zeros(num_easy_ham_train)

    X_train_hard_ham = make_data("hard_ham", 0, num_hard_ham_train,
vocab_dict)
    Y_train_hard_ham = np.zeros(num_hard_ham_train)
```

```
    X_train = np.concatenate((X_train_spam, X_train_easy_ham,
X_train_hard_ham), axis=0)
    Y_train = np.concatenate([Y_train_spam, Y_train_easy_ham,
Y_train_hard_ham])

    X_val_spam = make_data("spam", num_spam_train, 500, vocab_dict)
    Y_val_spam = np.ones(500 - num_spam_train)

    X_val_easy_ham = make_data("easy_ham", num_easy_ham_train, 1551,
vocab_dict)
    Y_val_easy_ham = np.zeros(1551 - num_easy_ham_train)

    X_val_hard_ham = make_data("hard_ham", num_hard_ham_train, 250,
vocab_dict)
    Y_val_hard_ham = np.zeros(250 - num_hard_ham_train)

    X_val = np.concatenate((X_val_spam, X_val_easy_ham, X_val_hard_ham),
axis=0)
    Y_val = np.concatenate([Y_val_spam, Y_val_easy_ham, Y_val_hard_ham])

    svm = sk.SVC(kernel='linear', C=1)
    svm.fit(X_train, Y_train)

    accuracy = svm.score(X_val, Y_val)

    print("\n{} training set:".format(name))
    print("   Correctly classified for C=1: {} %".format(round(accuracy *
100, 2)))

    svm = sk.SVC(kernel='linear', C=100)
    svm.fit(X_train, Y_train)

    accuracy = svm.score(X_val, Y_val)

    print("   Correctly classified for C=100: {} %".format(round(accuracy *
100, 2)))


def support_vector_machines():
    lineal_kernel()
    gaussian_kernel()
    parameter_selection()

def spam_detection():
    training_set(500, 1551, 0, "first")
    training_set(400, 1300, 150, "second")
    training_set(350, 1200, 250, "third")


def main():
    support_vector_machines()
    spam_detection()


main()
```