Práctica 7. Detección de Spam

Parte A. Support Vector Machines

1. Kernel lineal

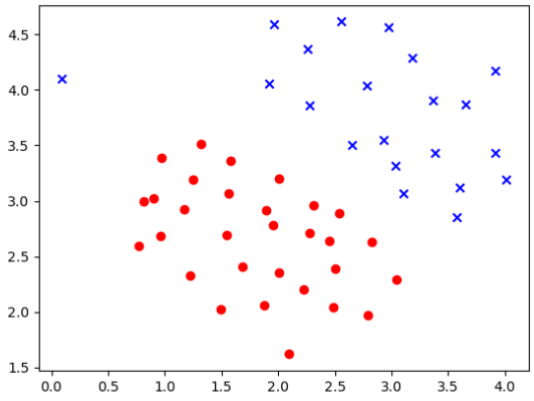
Para la carga del conjunto de datos se ha utilizado el método ***load_data()***:

```
def load_data(file):
    m = loadmat(file)
    X = m['X']
    y = m['y'].ravel()

    return X, y
```

Para empezar, se muestra la gráfica del primer conjunto de datos del fichero *ex6data1.mat*. En esta gráfica se aprecia una separación considerable en el conjunto de datos. Para mostrar la gráfica se ha utilizado el método ***print_data()***:

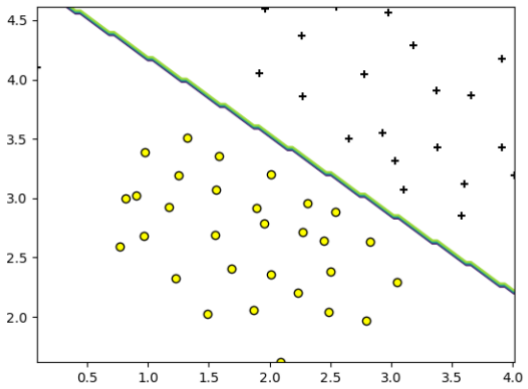
```
def print_data(X, y):
    x0 = (X[:, 0].min(), X[:, 0].max())
    x1 = (X[:, 1].min(), X[:, 0].max())
    p = np.linspace(x0, x1, 100)

    x1, x2 = np.meshgrid(p, p)
    zeros = (y == 0).ravel()
    ones = (y == 1).ravel()

    plt.figure()
    plt.scatter(X[ones, 0], X[ones, 1], color='b', marker='x')
    plt.scatter(X[zeros, 0], X[zeros, 1], color='r', marker='o')

    plt.show()
```

La gráfica es la que se muestra a continuación:

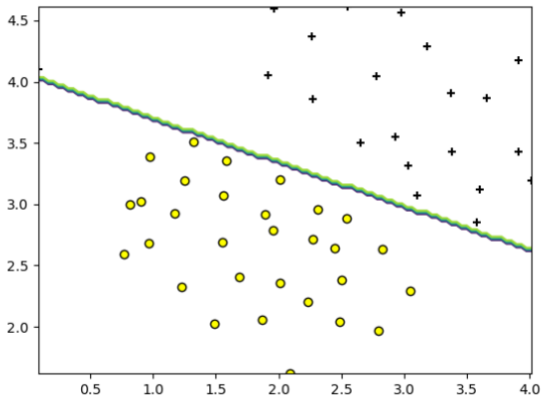


Utilizando la clase ***SVC*** proporcionada por la librería ***sklearn*** y especificando el kernel como lineal se obtienen, dependiendo del parámetro C las siguientes gráficas:

- Con C = 1:



- Con C = 100:



Como vemos en las gráficas, aunque ambas son capaces de diferenciar entre ambos tipos de manera perfecta, para C=1 podemos ver una separación con una mayor distancia entre la línea y cada uno de los puntos. Esto permite una mayor distribución a la hora de introducir nuevos datos.

El código para obtener este resultado es el siguiente:

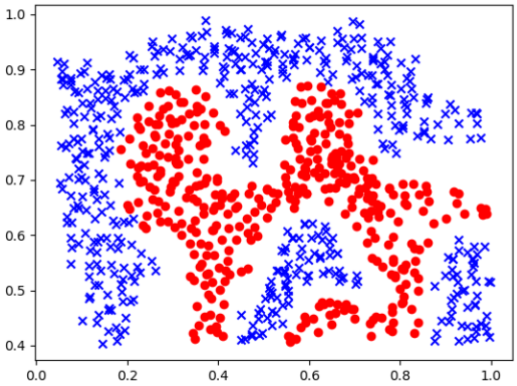
```
X, y = load_data(file='data/ex6data1.mat')
svm = skl.SVC(kernel='linear', C=1) # C = 100
svm.fit(X, y)
border_data(X, y, svm)
```

Y la definición del método ***border_data()*** es el siguiente:

```
def border_data(X, y, svm):
    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.show()
    plt.close()
```

2. Kernel Gaussiano

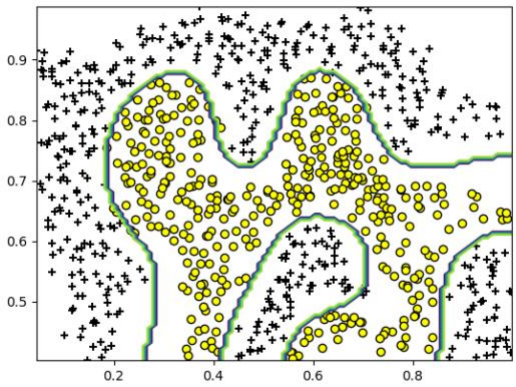
A continuación, se muestra la gráfica obtenida a partir del conjunto de datos proporcionado por el fichero *ex6data2.mat*.



El código para mostrar esta gráfica es el mismo utilizado en el apartado anterior. Para mostrar la gráfica donde se aprecia el uso del kernel gaussiano a partir de la clase SVC proporcionada por la librería ***sklearn*** es el siguiente:

```
c = 1
sigma = 0.1
svm = skl.SVC(kernel='linear', C = c, gamma = (1/(2 * sigma**2)))
svm.fit(X, y)
border_data(X, y, svm)
```

Y la gráfica resultante del método ***border_data()*** es:



Si se aplicase el kernel lineal el resultado sería una línea recta que no separaría de manera correcta los diferentes conjuntos de datos. Con el kernel gaussiano se puede apreciar que el modelo lo separa casi a la perfección.

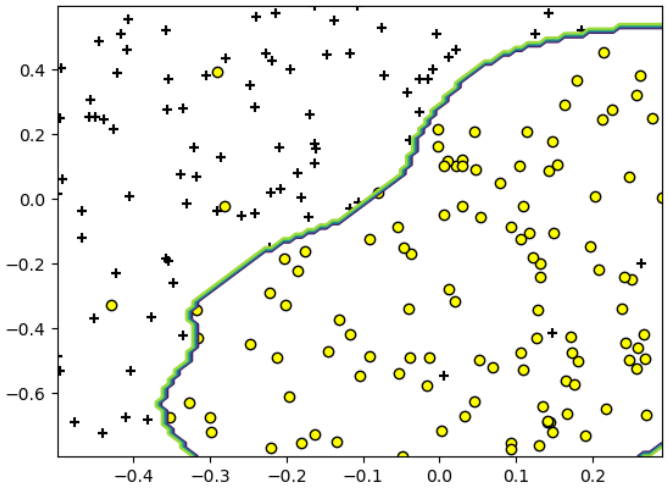
3. Elección de los parámetros C y σ

A continuación, se seleccionarán los valores de C y σ para un modelo de SVM con kernel gaussiano que clasifique el tercer conjunto de datos ***ex6data3.mat***. En este conjunto de datos, además de los datos de entrenamiento X e y, se incluyen datos de validación Xval e yval que servirán para evaluar el modelo aprendido.

Se ha definido el método ***main_chooseCAndSigma()***:

```python
def main_chooseCAndSigma():
    data = loadmat('data/ex6data3.mat')
    X = data['X']
    y = data['y'].ravel()
    x_val = data['Xval']
    y_val = data['yval'].ravel()
    scores = chooseCAndSigma(X, y, x_val, y_val)
    print(scores)
    ind = np.where(scores == scores.max())
    params = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    indp1 = ind[0][0]
    indp2 = ind[1][0]

    svm = skl.SVC(kernel='rbf', C = params[indp1], gamma = 1 / (2*params[indp2] ** 2))
    svm.fit(X, y)
    border_data(x_val, y_val, svm)
    print("C = {}, sigma = {}".format(params[indp1], params[indp2]))

    print(ind)
```

Como resultado se obtiene que el C óptimo es 1 y sigma = 0.1. La gráfica resultante es la siguiente:



El método para hallar estos parámetros es el siguiente:

```python
def chooseCAndSigma(X, y, x_val, y_val):
    params = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(params), len(params)))
    res = []

    for v in params:

        for sigma in params:
            svm = skl.SVC(kernel='rbf', C = v, gamma = 1 / (2*sigma ** 2))
            svm.fit(X, y)
            x_pred = svm.predict(x_val)
            correct = sum(x_pred == y_val) / x_pred.shape[0] * 100
            res.append(correct)
            scores[params.index(v), params.index(sigma)] = correct

    return scores
```

Detección de Spam

1. Lectura y procesamiento de los datos

Se ha procesado los conjuntos de datos conjuntos de datos de correo spam (spam.zip) y no spam (easy_ham.zip, más fáciles de identificar como correo no spam, y hard_ham.zip más fáciles de confundir con spam).
Se ha creado los siguientes métodos para conseguir esto:

```python
def transformAVector(vector, dictionary):
    email = np.zeros(1899)
    for word in vector:
        if word in dictionary.keys():
            email[dictionary.get(word) - 1] += 1
    return email

def load_examples(file, num_examples):
    response = np.zeros((num_examples, 1899))
    dictionary = getVocabDict()
    for i  in range(1, num_examples + 1):
        content = codecs.open('{}/{}.txt'.format(file, str(i).zfill(4)), encoding = 'utf', errors = 'ignore').read()
        vector = email2TokenList(content)
        transform = transformAVector(vector, dictionary)
        response[i - 1] = transform
    return response
```

Este código es parte de un proceso para cargar y transformar datos de correo electrónico en vectores numéricos, con el objetivo de analizar o entrenar modelos de aprendizaje automático. El proceso convierte el contenido de archivos de texto en una representación vectorial, usando un diccionario para mapear palabras a índices específicos y generar vectores de conteo. Estos vectores luego se pueden usar para análisis posteriores, como el entrenamiento de modelos de clasificación para detección de spam.

2. Entrenamiento y validación

El proceso para encontrar los valores óptimos de C y sigma se divide en dos funciones principales: ***findOptimalCAndSigma*** y ***train***.

- ***findOptimalCAndSigma***

Esta función busca los valores óptimos de C y sigma evaluando un conjunto de parámetros y seleccionando los que producen la mayor precisión en un conjunto de validación para luego urtilizarlos en el modelo SVM.

```python
def findOptimalCAndSigma(x_train, y_train, x_val, y_val):
    params = [ 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30 ]
    scores = np.zeros((len(params), len(params)))

    for C in params:
        print('Comprobación de C en {}...'.format(C))
        for sigma in params:
            print('Comprobación de Sigma en {}...'.format(sigma))
            svm = skl.SVC(kernel='rbf', C = C, gamma = 1/(2*sigma**2))
            svm.fit(x_train, y_train)
            accuracy = accuracy_score(y_val, svm.predict(x_val))
            print('Precisión con estos parámetros: {}'.format(accuracy))
            scores[params.index(C), params.index(sigma)] = accuracy

    optimalC = params[np.where(scores == scores.max())[0][0]]
    optimalSigma = params[np.where(scores == scores.max())[1][0]]
    print('Se ha determinado que el C óptimo es {}'.format(optimalC))
    print('Se ha determinado que el sigma óptimo es {}'.format(optimalSigma))
    return optimalC, optimalSigma
```

- ***train***

Esta función entrena un modelo SVM con los datos de correo electrónico proporcionados y utiliza ***findOptimalCAndSigma*** para determinar los mejores valores de C y sigma.

```python
def train(eham, hham, spam):
    yeham = np.zeros(2551)
    yhham = np.zeros(250)
    yspam = np.ones(500)
    random_state = 231052021

    eham_train , eham_test , yeham_train, yeham_test = train_test_split(eham,yeham, test_size = 0.25, random_state = random_state)
    hham_train , hham_test , yhham_train, yhham_test = train_test_split(hham,yhham, test_size = 0.25, random_state = random_state)
    spam_train, spam_test, yspam_train, yspam_test = train_test_split(spam,yspam, test_size = 0.25, random_state = random_state)

    x_train1 = np.concatenate((eham_train, hham_train, spam_train))
    y_train1 = np.concatenate((yeham_train, yhham_train, yspam_train))
    x_test = np.concatenate((eham_test, hham_test, spam_test))
    y_test = np.concatenate((yeham_test, yhham_test, yspam_test))

    xtrain, xval, ytrain, yval = train_test_split(x_train1, y_train1, test_size = 0.30, random_state = random_state)
    optimalC, optimalSigma = findOptimalCAndSigma(x_train = xtrain, y_train= ytrain, x_val = xval, y_val= yval)

    svmop = skl.SVC(kernel = 'rbf', C = optimalC, gamma = 1/(2*optimalSigma**2))
    svmop.fit(x_train1,y_train1)
    print(accuracy_score(y_test, svmop.predict(x_test)))
```

3. Resultados

Ejecutando estos métodos en la función ***main*** se ha llegado a la conclusión que los parámetros óptimos son C = 30 y sigma = 30, proporcionando una precisión del 0.9576271186440678