Práctica 7: Detección de spam

Material proporcionado

Archivo	Descripción
`data/ex6data1.mat`	Conjunto de datos 1
`data/ex6data2.mat`	Conjunto de datos 2
`data/ex6data3.mat`	Conjunto de datos 3
`data_spam/spam.zip`	Conjunto de datos con correo spam
`data_spam/easy_ham.zip`	Conjunto de datos con correo que no es spam
`data_spam/hard_ham.zip`	Conjunto de datos con correo que no es spam
`vocab.txt`	Lista de vocabulario
`utils.py`	Funciones auxiliares para leer el diccionario y hacer el procesamiento previo de un correo electrónico

Parte A: Support Vector Machines

El objetivo de la primera parte de la práctica es familiarizarse con el uso del clasificador SVM que incorpora sckit-learn, para luego aplicarlo en la segunda parte de la práctica.

Kernel lineal

La clase `sklearn.svm.SVC`[1] instancia un clasificador SVM utilizando el parámetro `C` de regularización, aplicando una función de kernel `kernel`, considerando que dos números en coma flotante son iguales si su diferencia es menor de `tol` y ejecutando como máximo `max_iter` pasadas del algoritmo de entrenamiento:

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', tol=0.001, max_iter=-1)
```

El clasificador se puede ejecutar luego sobre un conjunto de datos de entrenamiento `X`, etiquetados con un vector `y` de 0s y 1s como en este ejemplo:

```
svm = SVC(kernel='linear', C=1.0)
svm.fit(X, y)
```

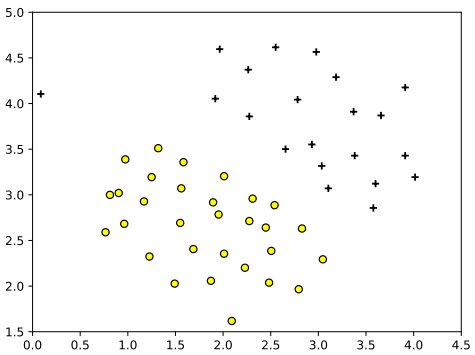

Figure 1.1: Conjunto de datos 1

En la Figura 1.1 se muestran los datos del primer conjunto de datos (`ex6data1.mat`), donde se aprecia que estos datos son linealmente separables. Lo primero que has de hacer es comprobar el efecto del parámetro C en el ajuste a los datos de entrenamiento utilizando el kernel lineal.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

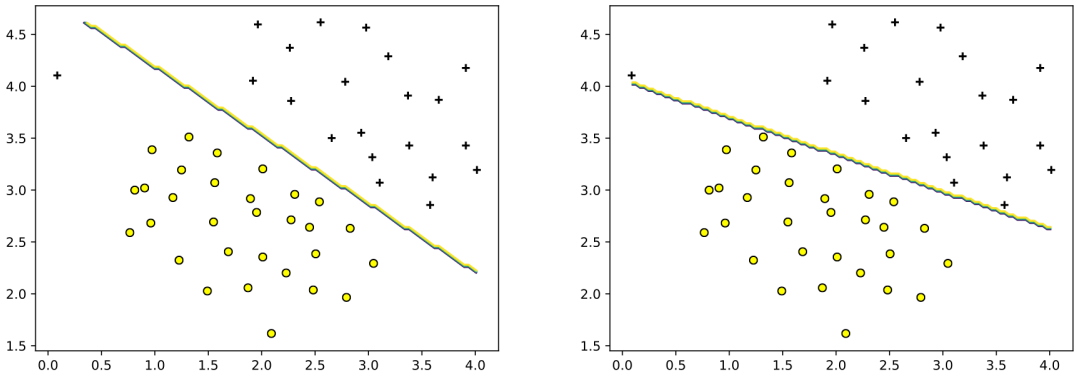

Figure 1.2: SVM con $C = 1$ (izquierda) y SVM con $C = 100$ (derecha)

En la Figura 1.2 se muestra el resultado del entrenamiento con $C = 1$ y con $C = 100$. En ambos casos has de utilizar la función `matplotlib.pyplot.contour` para visualizar la frontera de separación de las predicciones del modelo `svm` ajustado a los datos de entrenamiento `X` e `y`.

```
    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)

    plt.figure()
    plt.contour(x1, x2, yp)
```

Kernel gaussiano

A continuación has de utilizar el kernel gaussiano para así poder entrenar una SVM que clasifique correctamente el segundo conjunto de datos (`ex6data2.mat`) que, como se puede observar en la Figura 1.3, no es linealmente separable.

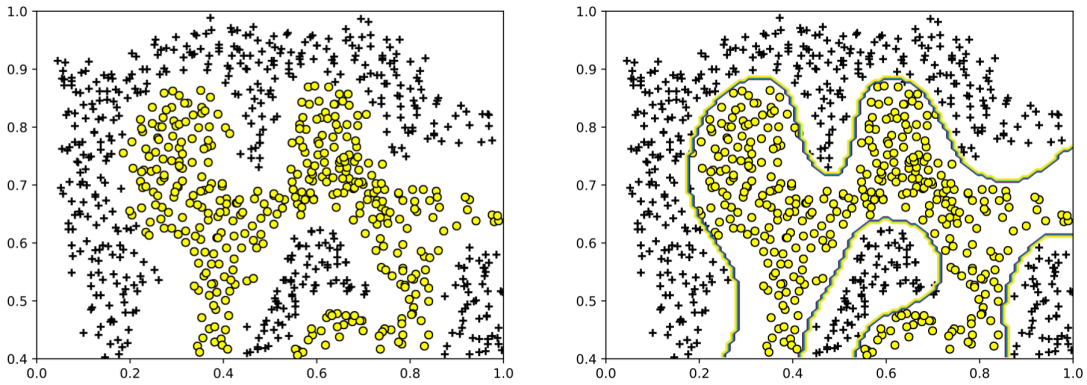

Figure 1.3: Datos no linealmente separables (izquierda) y SVM kernel gaussiano (derecha)

La función de kernel gaussiano calcula la distancia entre dos ejemplos de entrenamiento $(x^{(i)}, x^{(j)})$ de la siguiente forma:

$$K_{gauss}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right)$$
$$= \exp\left(-\gamma\|x^{(i)} - x^{(j)}\|^2\right)$$

En la Figura 1.3 se muestra la frontera no lineal definida por el modelo de SVM calculado con el kernel RBF (que es equivalente al gaussiano, sustituyendo la constante $1/2\sigma^2$ por γ) para `C = 1` y `sigma = 0.1`:

```
svm = SVC(kernel='rbf', C=C, gamma=1 / (2 * sigma**2))
```

Elección de los parámetros C y σ

A continuación, seleccionarás 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. Usando la función `scipy.io.loadmat` para leer el archivo `ex6data3.mat` obtendrás un diccionario que contiene las claves `X`, `y`, `Xval` e `yval`.

Has de generar modelos para C y σ tomando valores del conjunto {0.01, 0.03, 0.1, 0.3, 1, 3, 10,

30}[2], generando un total de $8^2 = 64$ modelos diferentes. Cada modelo lo debes evaluar sobre el conjunto de datos de validación `Xval` e `yval`, calculando el porcentaje de estos ejemplos que clasifica correctamente. En la Figura 1.4 se muestra la frontera de decisión para los valores de `C` y `sigma` que generan el modelo con menor porcentaje de error sobre los datos de validación.

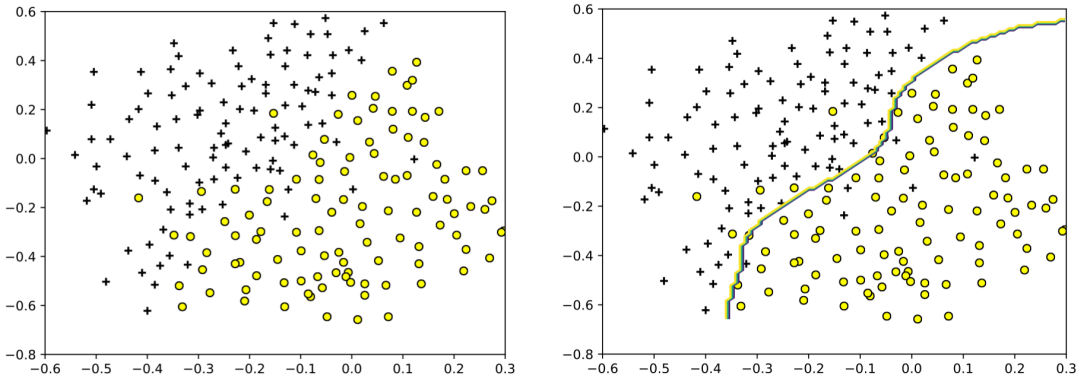

Figure 1.4: Datos no linealmente separables (izquierda) y SVM kernel gaussiano (derecha)

Detección de spam

Lectura y procesamiento de los datos

Con la práctica se proporcionan 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) extraídos del SpamAssassin Public Corpus[3], y tu objetivo será utilizarlos de la manera que creas más conveniente para generar y evaluar modelos que detecten el spam.

En primer lugar, deberás procesar los correos electrónicos para generar los datos de entrenamiento y validación. En los sistemas de detección de correo spam se suele hacer un procesamiento previo que transforma el texto de los mensajes para facilitar el proceso de aprendizaje. Con la práctica se proporciona la función `email2TokenList` que se encarga de:

- eliminar la cabecera del mensaje,
- pasar todo el texto a minúsculas,

[2]Puedes simplificar el cálculo de los valores si los aproximas empezando por 0.01 y multiplicando por 3 en cada iteración.

[3]https://spamassassin.apache.org/old/publiccorpus/

- eliminar las etiquetas HTML,
- normalizar las URLs, sustituyéndolas todas por el texto "`httpaddr`",
- normalizar las direcciones de correo, sustituyéndolas todas por el texto "`emailaddr`",
- sustituir todos los números por el texto "`number`",
- sustituir todas las apariciones del signo $ por el texto "`dollar`",
- sustituir cada palabra por su raíz, utilizando para ello el algoritmo de Porter que se importa del toolkit NLTK para procesamiento de lenguaje natural,
- eliminar todos los signos de puntuación.

De esta forma, leyendo un fichero del corpus y procesándolo con la función `email2TokenList`:

```
    email_contents = open( 'spam/0001.txt', 'r' ).read()
    email  = email2TokenList(email_contents)
```

pasamos de un texto como este:

```
...
Más líneas de cabecera
...
<CENTER>Save up to 70% on Life Insurance.</CENTER></FONT><FONT color=3D#ff=
0000
face=3D"Copperplate Gothic Bold" size=3D5 PTSIZE=3D"10">
<CENTER>Why Spend More Than You Have To?
<CENTER><FONT color=3D#ff0000 face=3D"Copperplate Gothic Bold" size=3D5 PT=
SIZE=3D"10">
<CENTER>Life Quote Savings
...
If you reside in any state which prohibits e-mail solicitations for insuran=
ce, please disregard this email.<BR>
</FONT><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR=
><BR><BR><BR></FONT></P></CENTER></CENTER></TR></TBODY></TABLE></CENTER></=
CENTER></CENTER></CENTER></BODY></HTML>
```

a una lista con estos elementos:

```
['save', 'up', 'to', 'number', 'on', 'life', 'insur', 'whi','spend',
'more', 'than', 'you', 'have', 'to', 'life', 'quot', 'save',
...
'pleas', 'disregard', 'thi', 'email']
```

Dado que los ficheros pueden contener caracteres que no estén codificados en UTF-8 podemos utilizar el módulo `codecs` para leer el fichero e ignorar los caracteres que no estén en UTF-8:

```
    email_contents = codecs.open( 'spam/0001.txt', 'r',
                                  encoding='utf-8', errors='ignore').read()
```

El siguiente paso es convertir el texto del mensaje en un vector de atributos. Para ello, primero es necesario seleccionar el conjunto de palabras que se usarán para describir los mensajes. Este conjunto viene dado en el archivo `vocab.txt` que contiene ordenadas alfabéticamente las 1899 palabras que aparecen al menos 100 veces en el corpus de mensajes. La función `getVocabDict` que se proporciona con la práctica se encarga de leer el fichero `vocab.txt` y devolver su contenido como un diccionario de python, donde las palabras del diccionario se usan como claves y se les asocia como valor el número de orden que ocupan en el diccionario, de 1 a 1899.

Cada correo electrónico se representa por un vector de 0s y 1s con $n = 1899$ componentes, una por cada palabra del vocabulario, de forma que la componente i-ésima del vector es 1 si la i-ésima palabra del vocabulario está en el mensaje y 0 si no está.

Entrenamiento de modelos

Entrena distintos sistemas de reconocimiento de spam, utilizando regresión logística, redes neuronales (tanto las que has implementado en la práctica 5 como las de Pytorch) y SVM, y compara sus resultados.

Submission of the assignment

The assignment must be delivered using the submission mechanism of the virtual campus. A single file will be delivered in pdf format containing the memory of the practice, including the code developed and the comments and graphs that are considered most appropriate to explain the results obtained.