Práctica 6: Support Vector Machines

Fecha de entrega: 18 de diciembre de 2014, 14.00h

Material proporcionado:

Fichero	Explicación
`ex6data1.mat`	Conjunto de datos 1.
`ex6data2.mat`	Conjunto de datos 2.
`ex6data3.mat`	Conjunto de datos 3.
`svmTrain.m`	Función que entrena una SVM.
`svmPredict.m`	Función que usa una SVM para hacer predicciones.
`plotData.m`	Dibuja datos en 2D.
`visualizeBoundaryLinear.m`	Dibuja una frontera de decisión lineal.
`visualizeBoundary.m`	Dibuja una frontera de decisión no lineal.
`linearKernel.m`	kernel lineal para una SVM.
`spam.zip`	Conjunto de datos con correo spam.
`easy_ham.zip`	Conjunto de datos con correo que no es spam.
`hard_ham.zip`	Conjunto de datos con correo que no es spam.
`vocab.txt`	Lista de vocabulario.
`getVocabList.m`	Carga la lista de vocabulario.
`porterStemmer.m`	Función de extracción de raíces.
`readFile.m`	Lee un archivo en una cadena de caracteres.
`processEmail.m`	Hace el procesamiento previo de un correo electrónico.

1. Support Vector Machines

El objetivo de la primera parte de la práctica es familiarizarse con el uso del clasificador SVM que se proporciona, para luego aplicarlo en la segunda parte de la práctica.

1.1. Kernel lineal

La función `svmTrain` proporcionada con la práctica calcula el modelo de una SVM a partir de un conjunto de datos de entrenamiento `X`, etiquetados con un vector `y` de 0s y 1s, utilizando el parámetro `C` de regularización, aplicando una función de kernel `kernelFunction`, considerando

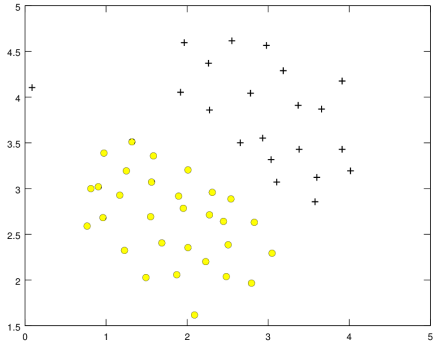
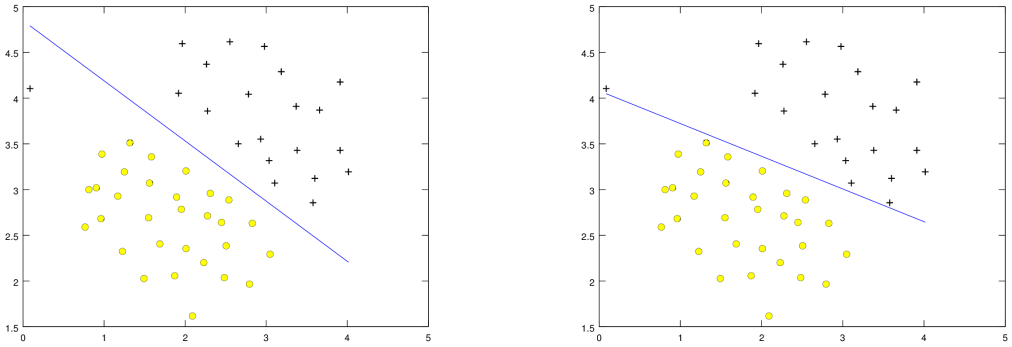


Figura 1: Conjunto de datos 1



(a) SVM con $C = 1$ (b) SVM con $C = 100$

Figura 2: Conjunto de datos 1

que dos números en coma flotante son iguales si su diferencia es menor de `tol` y ejecutando como máximo `max_passes`:

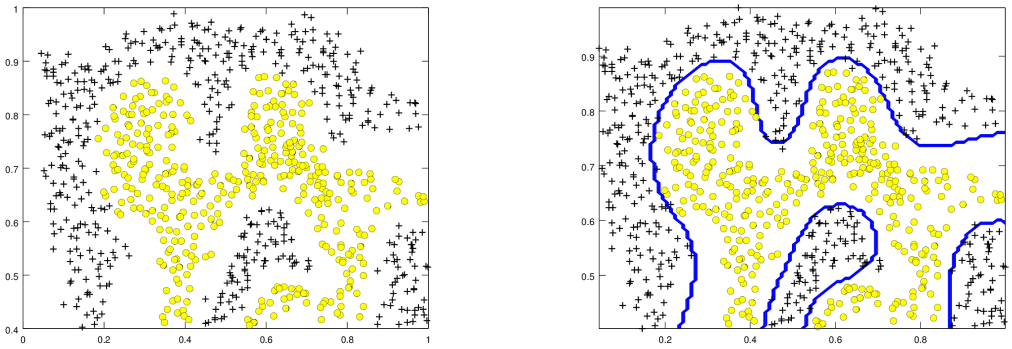
```
function [model] = svmTrain(X, Y, C, kernelFunction, tol, max_passes)
```

En la Figura 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 la función de kernel lineal (`linearKernel.m`) que se proporciona con la práctica:

```
model = svmTrain(X, y, C, @linearKernel, 1e-3, 20);
```

En la Figura 2a se muestra el resultado del entrenamiento con $C = 1$ y en la Figura 2b con $C = 100$. En ambos casos has de utilizar la función `visualizeBoundaryLinear` para visualizar la frontera de separación, a partir del modelo calculado por `svmTrain` y de los datos de entrenamiento `X` e `y`.



(a) Datos (b) SVM con kernel gaussiano

Figura 3: Conjunto de datos 2

1.2. Kernel gaussiano

A continuación has de implementar una función que calcule 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 3a, no es linealmente separable.

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

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

y has de implementarla en una función de Octave con la siguiente cabecera:

```
function sim = gaussianKernel(x1, x2, sigma)
```

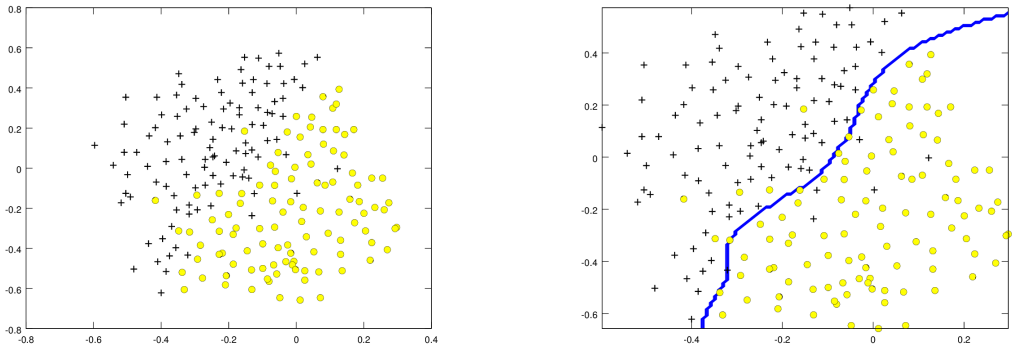
En la figura 3b se muestra la frontera no lineal definida por el modelo de SVM calculado con el kernel gaussiano para `C = 1` y `sigma = 0.1` y visualizada con la función `visualizeBoundary` que se proporciona con la práctica:

```
model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
visualizeBoundary(X, y, model);
```

1.3. 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. Has de generar modelos para C y σ tomando valores del conjunto 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30[1], generando un total de $8^2 = 64$ modelos diferentes.

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



(a) Datos (b) Kernel gaussiano

Figura 4: Conjunto de datos 3

Cada modelo lo debes evaluar sobre el conjunto de datos de validación `Xval` e `yval`, calculando el porcentaje de estos ejemplos que clasifica correctamente. Para ello puedes usar la función `svmPredict`, proporcionada con la práctica, que calcula el vector con las salidas de un modelo SVM para una matriz de ejemplos de entrenamiento dada.

2. Detección de spam

En esta segunda parte de la práctica utilizarás las funciones para el cálculo de modelos SVM que has probado en la primera parte para llevar a cabo experimentos en la detección de correo spam.

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[2], y tu objetivo será utilizarlos de la manera que creas más conveniente para generar y evaluar modelos SVM que detecten el spam.

En primer lugar, deberás procesar los correo 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 `processEmail` que se encarga de:

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

[2] `http://spamassassin.apache.org/publiccorpus`

- sustituir cada palabra por su raíz, utilizando para ello el algoritmo de Porter que se proporciona con la función `porterStemmer`,
- eliminar todos los signos de puntuación.

De esta forma, utilizando la función `readFile` incluida en la práctica para leer un fichero y devolver su contenido en una cadena y procesando el contenido del mensaje con la función `processEmail`:

```
file_contents = readFile('easy_ham/0024.txt');
email  = processEmail(file_contents);
```

pasamos de un texto como este:

```
...
Más líneas de cabecera
...
List-Id: Irish Linux Users' Group <ilug.linux.ie>
X-Beenthere: ilug@linux.ie

Can someone explain what type of operating system Solaris is... as ive never
seen or used it i dont know wheather to get a server from Sun or from DELL i
would prefer a linux based server and Sun seems to be the one for that but
im not sure if Solaris is a distro of linux or a completely different
operating system? can someone explain...

Kiall Mac Innes


--
Irish Linux Users' Group: ilug@linux.ie
http://www.linux.ie/mailman/listinfo/ilug for (un)subscription information.
List maintainer: listmaster@linux.ie
```

a este otro:

```
can someon explain what type of oper system solari is as iv never seen or
us it i dont know wheather to get a server from sun or from dell i would
prefer a linux base server and sun seem to be the on for that but im not
sure if solari is a distro of linux or a complet differ oper system can
someon explain kiall mac inn irish linux user group emailaddr httpaddr
for un subscript inform list maintain emailaddr
```

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 `getVocabList` que se proporciona con la práctica se encarga de leer el fichero `vocab.txt` y devolver su contenido como un array de celdas (*cell-array*) de Octave, que es el tipo de datos que permite definir una array de cadenas. Para acceder al elemento `i` del array de celdas `vocabList` se escribe `vocabList{i}` y `length(vocabList)` permite obtener su número de elementos.

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. Para comprobar si una palabra está o no en el vocabulario y conocer en qué posición, es conveniente, por razones de eficiencia, que transformes el array de celdas que devuelve `vocabList` en una estructura de Octave que tenga a las palabras del vocabulario como campos y su número de orden como valor.

3. Entrega de la práctica

La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicadas en la cabecera de la práctica.

Se entregará un único fichero en formato pdf que contenga la memoria de la práctica, incluyendo el código desarrollado y los comentarios y gráficas que se estimen más adecuados para explicar los resultados obtenidos.