Ap

APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 6

FDI-UCM

Iván Aguilera Calle – Daniel García Moreno

# Objetivo

El objetivo de esta séptima práctica es utilizar el clasificador SVM con dos kernels distintos, kernel lineal y kernel gaussiano. Una vez utilizados, se utilizarán para detectar correos considerados como “spam”. Para ello se utilizará tres tipos de datos adjuntos. Por un lado correos que se consideran spam. Por otro lado correos que no son considerados spam de manera sencilla. Y por último, una serie de correos que no son “spam” pero que es fácil categorizarlos mal.

# Implementación

En primer lugar, se practicará con los SVM a través de una serie de datos de ejemplo. Para ello, lo primero que debemos de realizar es visualizar los datos con los que nos estamos enfrentando. Para ello se ha ejecutado los comandos que podemos observar en la figura 1 y obtendremos la siguiente figura:

<

>> plotData(X, y)

Figura 1

Ahora calcularemos diferentes modelos de SVM con una función facilitada con la práctica, *svmTrain*.

Empezaremos calculando el modelo para el primer conjunto de datos (ex6data1). Para este conjunto de datos se usar el kernel “linealkernel”. Vamos a comprobar el efecto que tiene al utilizar distintos valores del parámetro “C” de la función *svmTrain*. Empezaremos con una valor de C = 1. Para ello, ejecutamos los comando que se pueden observar en la figura 2:

Figura 2

>> C = 1;

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

Una vez tenemos el modelo calculado, ejecutamos la función *visualizeBoundaryLinear*, proporcionada con la práctica, para visualizar la frontera de decisión que se obtiene con el modelo calculado anteriormente. Para ello, ejecutamos el comando de la figura 3 y obtenemos la siguiente gráfica:

>> visualizeBoundaryLinear(X, y, model);

Figura 3

Ahora repetiremos el paso anterior pero con un valor de C igual 100. Para ello ejecutamos los comandos que podemos observar en la figura 4 obteniendo la siguiente gráfica:

>> C = 100;

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

>> visualizeBoundaryLinear(X, y, model);

Figura 4

Ahora se va a calcular modelos de SVM utilizando un kernel gaussiano. Para ello, se va a utilizar un segundo conjunto de datos (ex6data2). Para este conjunto de datos es más apropiado utilizar este tipo de kernel, ya que no es linealmente separable.

El kernel gaussiano sigue la siguiente ecuación:

Este kernel lo hemos implementado en la función *gaussianKernel*, tal y como podemos observar en la figura 5:

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

sumatorio = sum((x1 - x2) .^ 2);

sim = exp(-((sumatorio)/(2 \* (sigma ^ 2))));

**end** **function**

Figura 5

Una vez implementado el kernel, utilizaremos la función *svmTrain* para calcular el modelo con C = 1 y sigma = 0.1, obteniendo la siguiente gráfica, tal y como podemos observar en la figura 6:

>> C = 1;

>> sigma = 0.1;

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

>> visualizeBoundary(X, y, model);

Figura 6

Para saber con qué valores de C y sigma entre un conjunto de valores obtenemos un mayor porcentaje de predicción correcta, se ha implementado la función *obtenerModelosGaussianos,* que genera diferentes modelos a partir de las combinaciones de un conjunto de valores de C y sigma. Su implementación la podemos observar en la figura 7:

**function** model = obtenerModelosGaussiano(X, y)

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

fila = 1;

**for** i = valores

**for** j = valores

model(fila) = svmTrain(X, y, i, @(x1 , x2) gaussianKernel (x1 , x2 , j));

fila++;

endfor

endfor

Figura 7

endfunction

A partir del grupo de modelos, se ha implementado la función *damePorcentaje,* que nos indica el porcentaje de buenas predicciones de cada modelo y nos indica con que valores de C y sigma se obtiene las mejores predicciones. Su implementación la podemos observar en la figura 8:

function p = damePorcentaje(models, Xval, yval)////CAMBIAR

m = columns(models);

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

fila = 1;

**for** i = 1:m

bienPredecidos = length(find((svmPredict(models(i), Xval) == yval) == 1));

p(i) = (bienPredecidos/rows(yval)) \* 100;

endfor

**for** i = valores

**for** j = valores

fprintf(['C: %f; sigma: %f; porcentaje: %f\n'], i, j, p(fila));

fila++;

endfor

endfor

[maximo ind] = max(p);

i = ceil(ind/columns(valores));

j = ind - ((i - 1) \* columns(valores));

fprintf(['Valor optimo->C: %f; sigma: %f; porcentaje: %f\n'], valores(i), valores(j), maximo);

Figura 8

endfunction