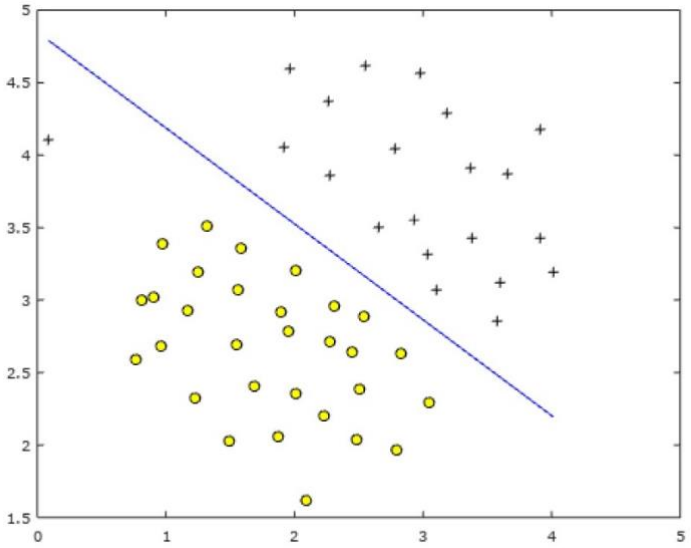
Práctica 6

Pablo Arranz Ropero
Juan Alberto Camino Sáez
Grupo 2

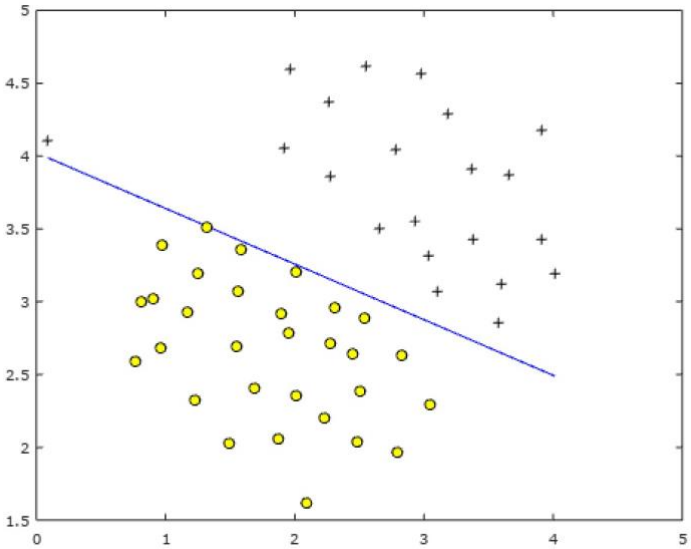
Práctica 6: Support Vector Machines

En esta práctica nos familiarizaremos con el uso del clasificador SVM para aplicarlo en la segunda parte de la práctica.

En primer lugar, hemos aplicado SVM para comprobar el efecto del parámetro C. En las siguientes gráficas se observa la separación aplicando distintos valores de C sobre unos datos linealmente separables. En primer lugar, con C = 1



En el siguiente caso, con C = 100:

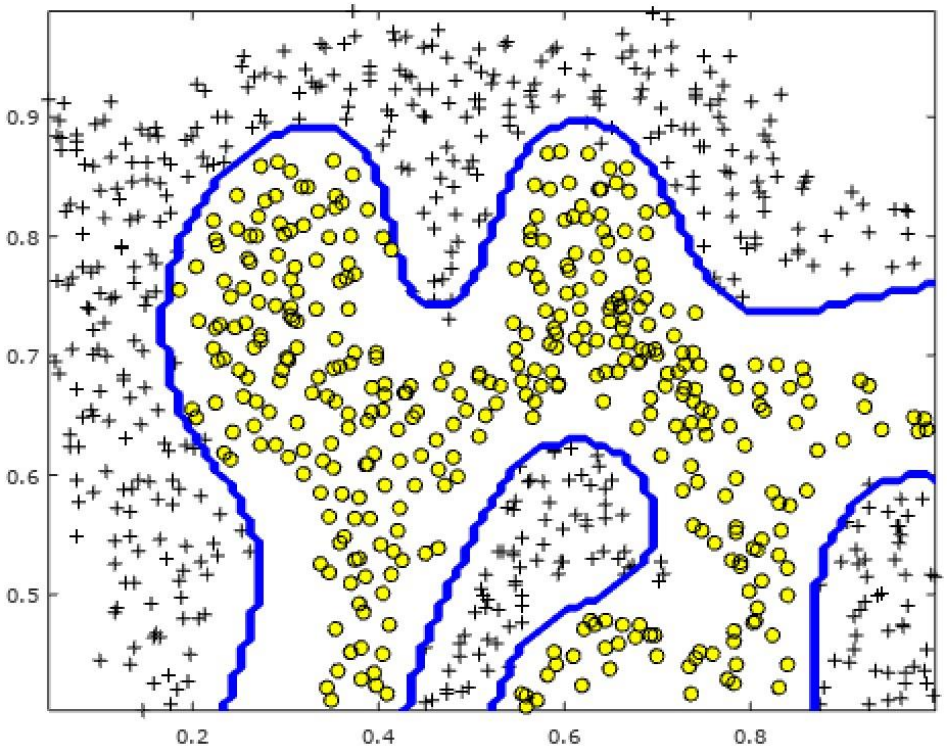


KERNEL GAUSSIANO

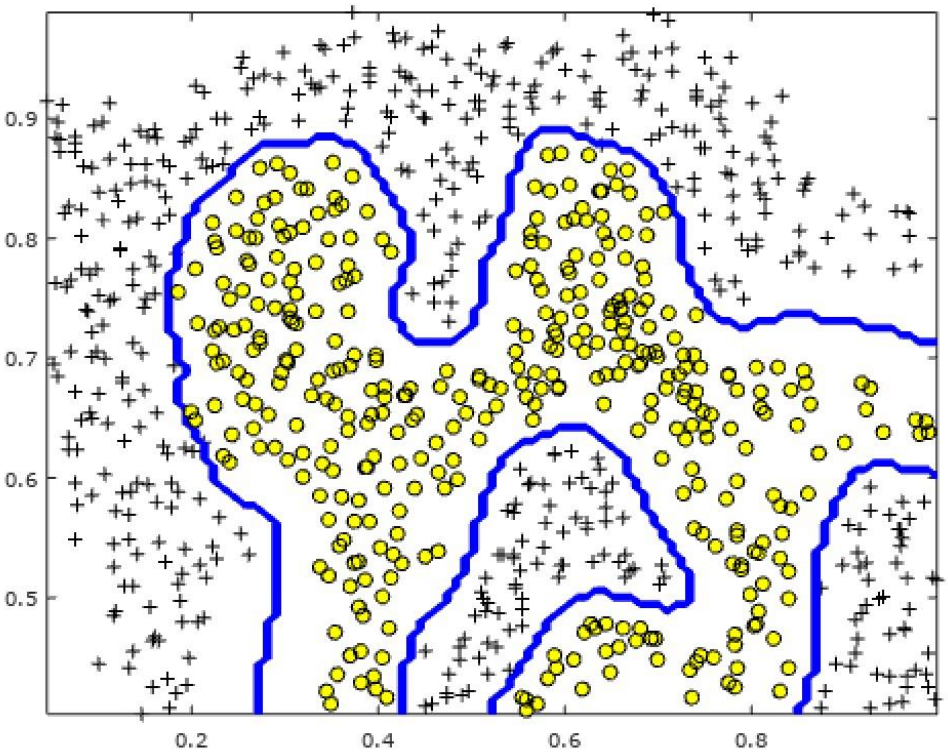
A continuación, procedemos a realizar un kernel Gaussiano implementando la función *gaussianKernel.m*, al que se le pasan x1, x2 y sigma y te devuelve el kernel Gaussiano entre los dos puntos.
Este el código de la función:

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

  sim = exp(-(sum((x1-x2).^2)/(2*(sigma^2))));

endfunction
```

Además, tras aplicar SVM con el kernel Gaussiano, nos presenta la siguiente separación en la gráfica. Usando C = 1 y $\sigma = 0.1$:



En el siguiente caso. Usamos C = 1 y $\sigma = 0.03$. Se puede observar un poco más de overfitting:

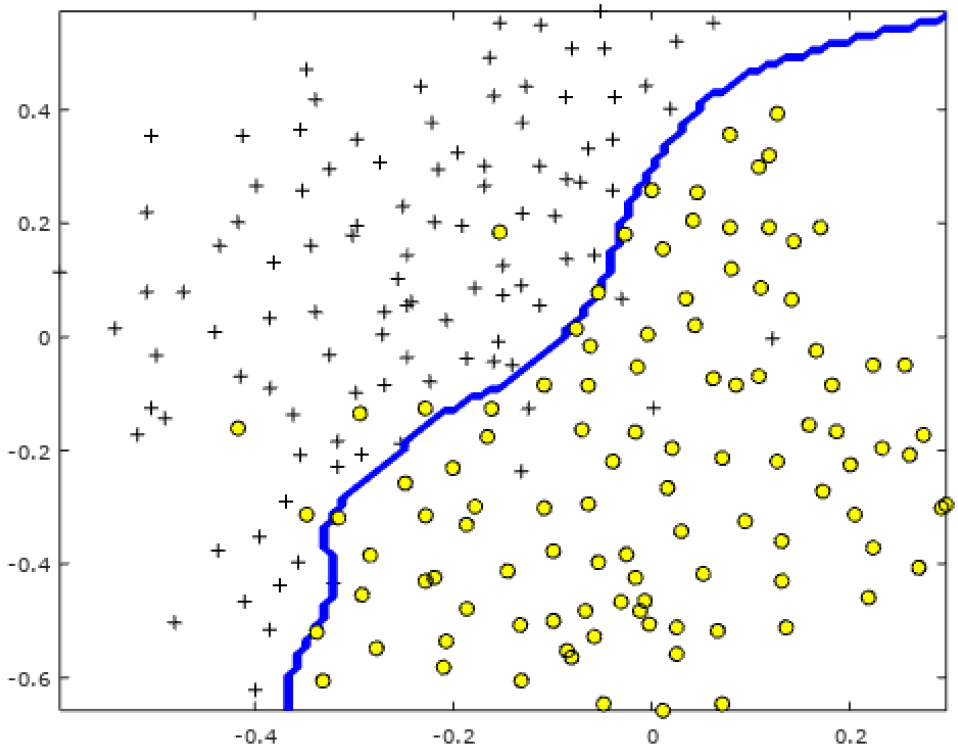


ELECCIÓN DE LOS PARÁMETROS C Y σ

En la siguiente parte de la práctica hemos creado una función llamada *chooseCandsigma.m*, al que le pasamos los datos de entrenamiento, de validación, el inicio de C y σ y el número de pasos (veces que hay que multiplicar por 3 C y σ), y se encarga de calcular la similitud de todos los modelos diferentes con los datos de validación y devuelve el mejor C y el mejor σ. Este es el código:

```
function [bestC, bestsigma] = chooseCandsigma(X, y, Xval, yval)

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

bestnumcorrect = 0;

for i = 1:columns(vect)
  C = vect(i);

  for j = 1:columns(vect)
    sigma = vect(j);

    printf('Step %d', ((i - 1) * columns(vect)) + j);

    model = svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
    numcorrect = sum(svmPredict(model, Xval) == yval)

    if (numcorrect > bestnumcorrect)
      bestC = C;
      bestsigma = sigma;
      bestnumcorrect = numcorrect;
    endif

  endfor

endfor

model = svmTrain(X, y, bestC, @(x1, x2) gaussianKernel(x1, x2, bestsigma));
visualizeBoundary(X, y, model);

endfunction
```

Los mejores valores de C y σ calculados son 1 y 0.1 respectivamente. Las últimas líneas de nuestra función pintan la gráfica usando el modelo elegido y los datos proporcionados quedando lo mostrado en la siguiente imagen:



DETECCIÓN DE SPAM

A continuación utilizaremos una serie de ficheros con correos que son spam y no spam y aplicaremos SVM para crear un modelo que indique si un correo es spam o no.

Hemos creado una función llamada *saveContentToMat.m,* la cual se encarga de leer todos los correos, de procesar cada email para comprobar qué palabras típicas de correos de spam aparecen en los mismos (creando para ello un vector de 0 y 1). Pór último, creamos el vector y, poniendo un 0 si el correo leído no es spam y un 1 si el correo leído es spam, y guardamos todos estos datos en un fichero, los de easyham (correos que no son spam y que son fáciles de identificar) en el fichero *easyham.mat*, los de hardham (correos que no son spam y que son difíciles de identificar) en un fichero *hardham.mat* y los de spam (correos que son spam) en un fichero *spam.mat.*
El usar tres ficheros es para acelerar los tiempos de carga y seguir teniendo diferenciados los correos de spam y no spam.

Este es el código de la función que se ha implementado:

```
function saveContentToMat()
  vocabList = getVocabList();

  for i = 1:2551
    if (i > 0 && i < 10)
      file_contents = readFile(strcat('easy_ham/000',num2str(i),'.txt'));
    elseif (i > 9 && i < 100)
      file_contents = readFile(strcat('easy_ham/00',num2str(i),'.txt'));
    elseif (i > 99 && i < 1000)
      file_contents = readFile(strcat('easy_ham/0',num2str(i),'.txt'));
    elseif (i > 999 && i < 10000)
      file_contents = readFile(strcat('easy_ham/',num2str(i),'.txt'));
    endif

    email = processEmail(file_contents);

    email_splited = strsplit(email);

    email_as_vector = ismember(vocabList, email_splited);

    easy_ham(i, :) = email_as_vector;
    printf(strcat(num2str(i),'/3301 \n'));
    fflush(stdout);

  endfor

  easy_ham(:, columns(easy_ham) + 1) = 0;

  save easy_ham.mat;
  clear;
  vocabList = getVocabList();

  for i = 1:250
    if (i > 0 && i < 10)
      file_contents = readFile(strcat('hard_ham/000',num2str(i),'.txt'));
    elseif (i > 9 && i < 100)
      file_contents = readFile(strcat('hard_ham/00',num2str(i),'.txt'));
    elseif (i > 99 && i < 1000)
      file_contents = readFile(strcat('hard_ham/0',num2str(i),'.txt'));
    endif

    email = processEmail(file_contents);

    email_splited = strsplit(email);

    email_as_vector = ismember(vocabList, email_splited);

    hard_ham(i, :) = email_as_vector;
    printf(strcat(num2str(2551 + i),'/3301 \n'));
    fflush(stdout);

  endfor

  hard_ham(:, columns(hard_ham) + 1) = 0;

  save hard_ham.mat;
  clear;
  vocabList = getVocabList();

  for i = 1:500
    if (i > 0 && i < 10)
      file_contents = readFile(strcat('spam/000',num2str(i),'.txt'));
    elseif (i > 9 && i < 100)
      file_contents = readFile(strcat('spam/00',num2str(i),'.txt'));
    elseif (i > 99 && i < 1000)
      file_contents = readFile(strcat('spam/0',num2str(i),'.txt'));
    endif

    email = processEmail(file_contents);

    email_splited = strsplit(email);

    email_as_vector = ismember(vocabList, email_splited);

    spam(i, :) = email_as_vector;
    printf(strcat(num2str(2551 + 250 + i),'/3301 \n'));
    fflush(stdout);
  endfor

  spam(:, columns(spam) + 1) = 1;

  save spam.mat;
  clear;

endfunction
```

A continuación, aplicamos SVM para clasificar estos correos, usando un 60% de los correos para el entrenamiento, un 20% para la validación y el 20% restante para test, ordenándolos de forma aleatoria.

Este es el código:

```
function [X, y, Xval, yval, Xtest, ytest] = loadandrandomize()

  %Cargamos easy_ham y lo reordenamos de manera aleatoria
  load('easy_ham.mat');
  [m_easy_ham,n] = size(easy_ham)
  idx = randperm(m_easy_ham);
  easy_ham_rand = easy_ham;
  easy_ham_rand(idx, :) = easy_ham;

  %Cargamos hard_ham y lo reordenamos de manera aleatoria
  load('hard_ham.mat');
  [m_hard_ham,n] = size(hard_ham)
  idx = randperm(m_hard_ham);
  hard_ham_rand = hard_ham;
  hard_ham_rand(idx, :) = hard_ham;

  %Cargamos spam y lo reordenamos de manera aleatoria
  load('spam.mat');
  [m_spam,n] = size(spam)
  idx = randperm(m_spam);
  spam_rand = spam;
  spam_rand(idx, :) = spam;

  datatrain_easy_ham = easy_ham_rand(1:floor(0.6*m_easy_ham),:);
  datacrossvalidation_easy_ham = easy_ham_rand(floor(0.6*m_easy_ham)+1:floor(0.8*m_easy_ham),:);
  datatest_easy_ham = easy_ham_rand(floor(0.8*m_easy_ham)+1:end,:);

  datatrain_hard_ham = hard_ham_rand(1:floor(0.6*m_hard_ham),:);
  datacrossvalidation_hard_ham = hard_ham_rand(floor(0.6*m_hard_ham)+1:floor(0.8*m_hard_ham),:);
  datatest_hard_ham = hard_ham_rand(floor(0.8*m_hard_ham)+1:end,:);

  datatrain_spam = spam_rand(1:floor(0.6*m_spam),:);
  datacrossvalidation_spam = spam_rand(floor(0.6*m_spam)+1:floor(0.8*m_spam),:);
  datatest_spam = spam_rand(floor(0.8*m_spam)+1:end,:);

  X = [datatrain_easy_ham ; datatrain_hard_ham ; datatrain_spam];
  y = X(:, end);
  X(:, end) = [];

  Xval = [datacrossvalidation_easy_ham ; datacrossvalidation_hard_ham ; datacrossvalidation_spam];
  yval = Xval(:, end);
  Xval(:, end) = [];

  Xtest = [datatest_easy_ham ; datatest_hard_ham ; datatest_spam];
  ytest = Xtest(:, end);
  Xtest(:, end) = [];

endfunction
```

Por último, aplicamos SVN sobre estos datos para ver su eficacia.