Práctica 6: Support Vector Machines

Grupo 13:
David Ortiz Fernández.
Andrés Ortiz Loaiza.

1. Support Vector Machines

En esta primera parte de la práctica, se hace una introducción a SVM. Se ha comenzado comprobando como afecta el parámetro C a una SVM con kernell lineal y usando unos datos linealmente separables.
A continuación, se ha desarrollado un kernell gausiano y se han usado para comprobar su uso unos valores de C y sigma de 1 y de 0.1 respectivamente, en la gráfica obtenida se ha podido ver como se creaba una frontera de separación correcta en el calificador.
Po ultimo hemos calculado el porcentaje de acierto del modelo: 98.841251, el cual nos indica la calidad de nuestro clasificador para unos datos no linealmente separables. Seguidamente hemos creado una función para seleccionar los valores de C y sigma óptimos, tras lo que hemos obtenido un acierto de: 94.786730.

Script principal

```
clear ;
close all;
%1. Flujo de datos:Support Vector Machines
%Visualizamos los datos
load('ex6data1.mat');

% Plot datos ex6data1.mat
% Plot para apreciar que los datos son linealmente separables
plotData(X, y);

printf('Presione Enter para continuar.\n');
pause;
%1.1. Kernel lineal
    %Probar con C=1 y C=100
    C = 1;
    %C = 100;
    model = svmTrain(X, y, C, @linearKernel, 1e-3, 20);
    %visualizar la frontera de separación
    visualizeBoundaryLinear(X, y, model);

    p = svmPredict(model, X);
    printf('Porcentaje de acierto del modelo: %f\n',mean(double(p == y)) * 100);

    printf('Presione Enter para continuar.\n');
    pause;

%1.2. Kernel gaussiano
    load('ex6data2.mat');

    % Plot para apreciar que los datos no son linealmente separables
    % Plot datos ex6data2.mat
    plotData(X, y);

    C = 1; sigma = 0.1;

    %entrenamiento del modelo
    model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
    %visualizar la frontera de separación
    visualizeBoundary(X, y, model);

    p = svmPredict(model, X);
    printf('Porcentaje de acierto del modelo: %f\n',mean(double(p == y)) * 100);

    fprintf('Presione Enter para continuar.\n');
    pause;

%1.3. Elección de los parámetros C y s
    load('ex6data3.mat');
    % Plot ex6data3.mat
    plotData(X, y);

    printf('Presione Enter para continuar.\n');
    pause;

    %calculo de los parametros C y sigma
    [C, sigma] = calculoCSigma(X, y, Xval, yval);

 %entrenamiento del modelo
model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
visualizeBoundary(X, y, model);

p = svmPredict(model, X);
printf('Porcentaje de acierto del modelo: %f\n',mean(double(p == y)) * 100);
```

Funciones :

gaussianKernel

```
%función que calcule el kernel gaussiano para así poder entrenar una SVM que
%clasifique correctamente el segundo conjunto de datos
function sim = gaussianKernel(x1, x2, sigma)
  x1 = x1(:); x2 = x2(:);

  %la función de kernel gaussiano calcula la distancia entre dos ejemplos de
  %entrenamiento (x(i), x(j))
  sim = 0;
  sim = exp(-1*(x1-x2)'*(x1-x2)/(2*sigma*sigma));

endfunction
```

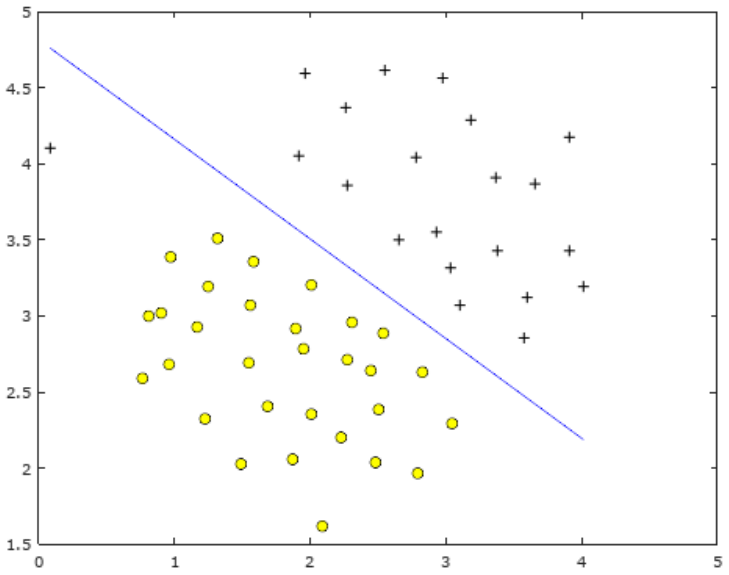
calculoCSigma

```
%funcion para el calculo de los valores de C y sigma y maximizan los valores
%bien clasificados
function [C, sigma] = calculoCSigma(X, y, Xval, yval)
minerror = inf;
values = [0.01 0.03 0.1 0.3 1 3 10 30];
%doble for para los 64 modelos diferentes
for Caux = values
  for sigmaAux = values
    printf('Entrenamiento y validacion para los valores de: \n[Caux, sigmaAux]
            = [%f %f]\n', Caux, sigmaAux);
   %culando el porcentaje de estos ejemplos que clasificaca correctamente
    model = svmTrain(X, y, Caux, @(x1, x2) gaussianKernel(x1, x2, sigmaAux));

   %calculo de los que se clasifican mal
    error = mean(double(svmPredict(model, Xval) ~= yval));
    printf('Prediccion de error: %f\n', error);

    %actualizacion del error minimo en caso de haber encontado uno menor
    if( error <= minerror )
      sigma = sigmaAux;
      C = Caux;
      minerror = error;
    endif
    printf('Valores de C = %f , sigma = %f con un error de prediccion de:  %f\n'
              , C, sigma, minerror);
    printf('-------------------------------------------------------------\n');
  endfor
endfor
endfunction
```

Resultados:

1.1 Kernel lineal

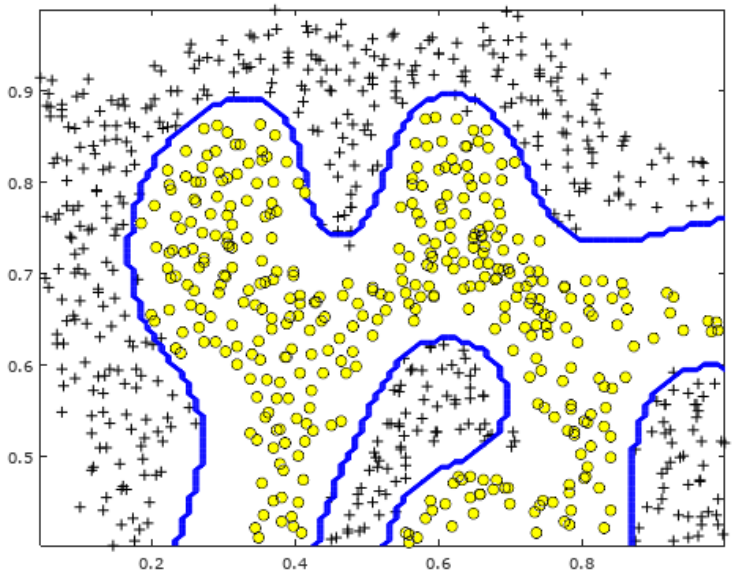
La siguiente gráfica muestra la frontera lineal definida por el modelo de SVM calculado con el kernel lineal para C = 1



Porcentaje de acierto del modelo: 98.039216

1.2 Kernel gaussiano

La siguiente gráfica muestra la frontera no lineal definida por el modelo de SVM calculado con el kernel gaussiano para C = 1 y sigma = 0.1

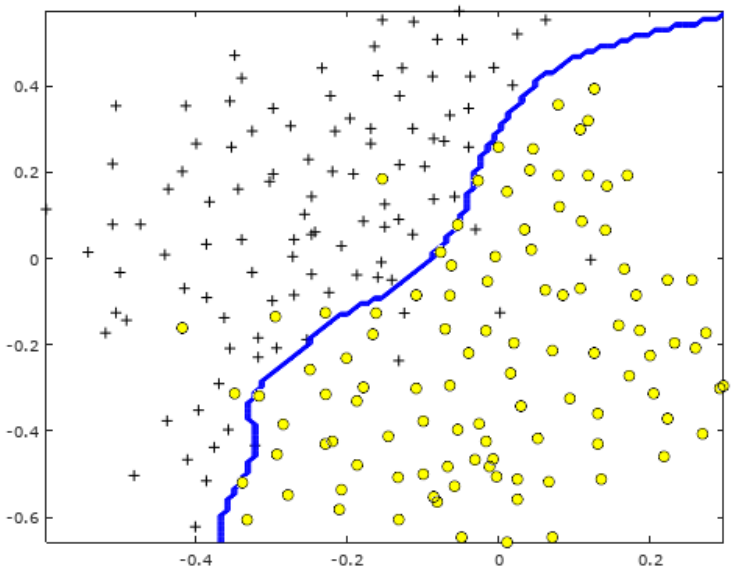


Porcentaje de acierto del modelo: 98.841251

.

1.3 Kernel gaussiano

La siguiente gráfica muestra la frontera no lineal definida por el modelo de SVM calculado con el kernel gaussiano para C y sigma calculados tomando valores del conjunto 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, generando así un total de 8 2 = 64 modelos diferentes para su cálculo.



Porcentaje de acierto del modelo: 94.786730

2. Detección de spam

En esta segunda parte nos hemos dispuesto usar una SVM para clasificar correos como spam y no spam. Hemos creado un script que lee todos los ficheros, los procesa y crea gracias a un diccionario un vector con las palabras típicas que identifican correos con spam. Con esto, la función crea 3 ficheros que contienen todos estos datos, un fichero para conjunto, easyham.mat, hardham.mat y spam.mat.

Tras esto hemos aplicado los conocimientos de la primera parte de la práctica y realizar así predicciones, para ello dividiendo los archivos por cross-validation. Este es el código de la función que se ha implementado:

Script principal

```
clear ;
close all;
warning('off','all');

%Cargamos easy_ham
load('easy_ham.mat');
[m_easy_ham,n] = size(easy_ham);

%Cargamos hard_ham
load('hard_ham.mat');
[m_hard_ham,n] = size(hard_ham);

%Cargamos spam
load('spam.mat');
[m_spam,n_spam] = size(spam);

%elegimos porcentajes del 60%,20% y 20% de CADA archivo
datatr_easy_ham = easy_ham(1:floor(0.6*m_easy_ham),:);
datacv_easy_ham = easy_ham(floor(0.6*m_easy_ham)+1:floor(0.8*m_easy_ham),:);
datatest_easy_ham = easy_ham(floor(0.8*m_easy_ham)+1:end,:);

datatr_hard_ham = hard_ham(1:floor(0.6*m_hard_ham),:);
datacv_hard_ham = hard_ham(floor(0.6*m_hard_ham)+1:floor(0.8*m_hard_ham),:);
datatest_hard_ham = hard_ham(floor(0.8*m_hard_ham)+1:end,:);

datatr_spam = spam(1:floor(0.6*m_spam),:);
datacv_spam = spam(floor(0.6*m_spam)+1:floor(0.8*m_spam),:);
datatest_spam = spam(floor(0.8*m_spam)+1:end,:);
```

```
X_tr = [datatr_easy_ham ; datatr_hard_ham ; datatr_spam];
%y es la ultima fila de X
y_tr = X_tr(:, end);
%quitamos y de X
X_tr=X_tr(:,1:end-1);
X_cv = [datacv_easy_ham ; datacv_hard_ham ; datacv_spam];
%y es la ultima fila de X
y_cv = X_cv(:, end);
%quitamos y de X
X_cv=X_cv(:,1:end-1);
X_test = [datatest_easy_ham ; datatest_hard_ham ; datatest_spam];
%y es la ultima fila de X
y_test = X_test(:, end);
X_test=X_test(:,1:end-1);
C = 0.1;
model = svmTrain(X_tr, y_tr, C, @linearKernel);
%Evaluacion del modelo
p = svmPredict(model, X_test);
printf('Porcentaje de acierto del modelo con kernel linial: %f\n',
                                                mean(double(p == y_test)) * 100);
%entrenamiento con kernell gausiano
C = 1;
sigma = 0.1;
[C, sigma] = calculoCSigma(X_tr, y_tr, X_cv, y_cv);
model= svmTrain(X_tr, y_tr, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
%evaluacion del modelo
p = svmPredict(model, X_tr);
printf('Porcentaje de acierto del modelo con kernel gausiano: %f\n',
mean(double(p == y_tr)) * 100);
```

Generación archivos

```
vocabList = getVocabList();
%flujo de creación de los dataset
%Añadimos ejemplos que son spam al dataset y los categorizamos como tal

%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

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

    email = processEmail(file_contents);

    email_splited = strsplit(email);
    email_as_vector = zeros(rows(vocabList), 1);
    email_as_vector = ismember(vocabList, email_splited);

    spam(j, :) = email_as_vector;
    printf(strcat(num2str(j),'/3301 \n'));
  endfor
%añadimos un 1 a cada fila para indicar que es spam
    spam(:, columns(spam) + 1) = 1;
    save spam.mat;
```

```
 vocabList = getVocabList();
%flujo de creación del dataset de entrenamiento
%Añadimos ejemplos que son spam al dataset y los categorizamos como tal

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


    email  = processEmail(file_contents);

    email_sp = strsplit(email);
    email_vec = zeros(rows(vocabList), 1);
    email_vec = ismember(vocabList, email_sp);

    spam(j, :) = email_vec;
    printf(strcat(num2str(250 + j),'/3301 \n'));
  endfor
%añadimos a  cada fila un 1 indicando que es spam
    hard_ham(:, columns(spam) + 1) = 1;
```

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

    email = processEmail(file_contents);

    email_sp = strsplit(email);
    email_vec = zeros(rows(vocabList), 1);
    email_vec = ismember(vocabList, email_sp);

    easy_ham(j, :) = email_vec;
    printf(strcat(num2str(750 + j),'/3301 \n'));
  endfor
%añadimos a cada fila un 0 indicando que no es spam
easy_ham(:, columns(easy_ham) + 1) = 0;
save easy_ham.mat;
printf('Presiones Enter para finalizar.\n');
pause;
```