Aprendizaje automático

Práctica 6

Miguel Ascanio Gómez

Esther Ávila Benito

En esta práctica se ha estudiado el uso del clasificador *Support Vector Machines* y lo hemos aplicado a un ejemplo de detección de correo *spam*.

# SUPPORT VECTOR MACHINES

## KERNEL LINEAL

clear;

load ex6data1.mat;

C = 1;

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

visualizeBoundaryLinear(X,y,model);

title("C = 1");

print("-dpng", "Parte1\_C1.png");

C = 100;

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

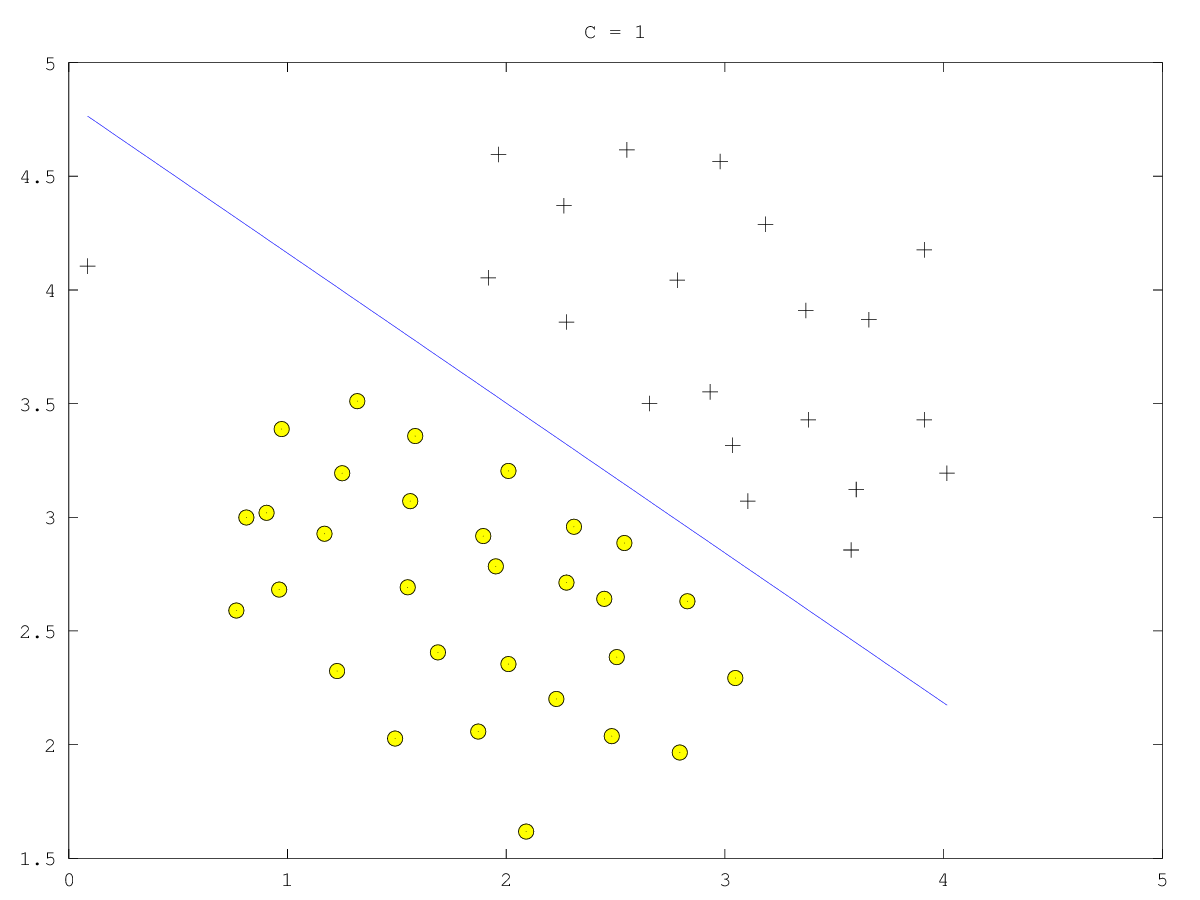
visualizeBoundaryLinear(X,y,model);

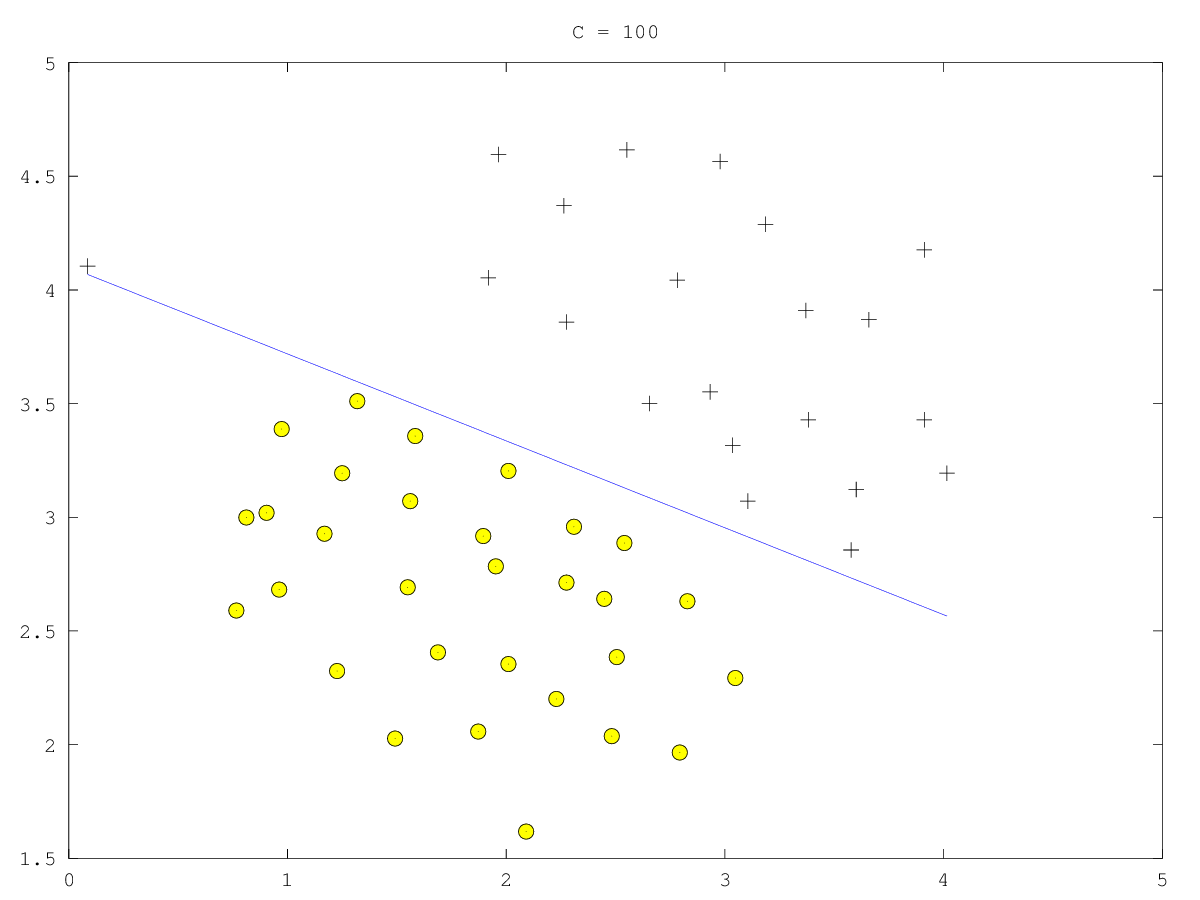
title("C = 100");

print("-dpng", "Parte1\_C100.png");

load("ex3data1.mat");

m = size(X, 1);





## KERNEL GAUSSIANO

Cálculo del *kernel gaussiano* con el que se entrenara una SVM para la clasificación de los datos contenidos en *ex6data2.mat*.

function sim = gaussianKernel(x1, x2, sigma)

sumandos = (x1 .- x2).^2;

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

endfunction

Se emplea el kernel gaussiano con *C* = 1 y *sigma* = 0.1

clear;

load ex6data2.mat;

C = 1;

sigma = 0.1;

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

visualizeBoundary(X,y,model);

## elección de los parámetros *C* y *sigma*

Se seleccionan los valores de *C*  *sigma* para la clasificación de los datos contenidos en ex6data3.mat. El cálculo de estos parámetros se realiza para los valores del conjunto 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30.

load ex6data3.mat;

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

i = 1;

j = 1;

porcentajes = [];

for C = valores

i

j = 1;

for sigma = valores

j

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

ypred = svmPredict(model, Xval);

dataSave(i,j).C = C;

dataSave(i,j).sigma = sigma;

dataSave(i,j).model = model;

dataSave(i,j).porcentaje = size(find(yval == ypred))(1) / size(yval)(1) \* 100;

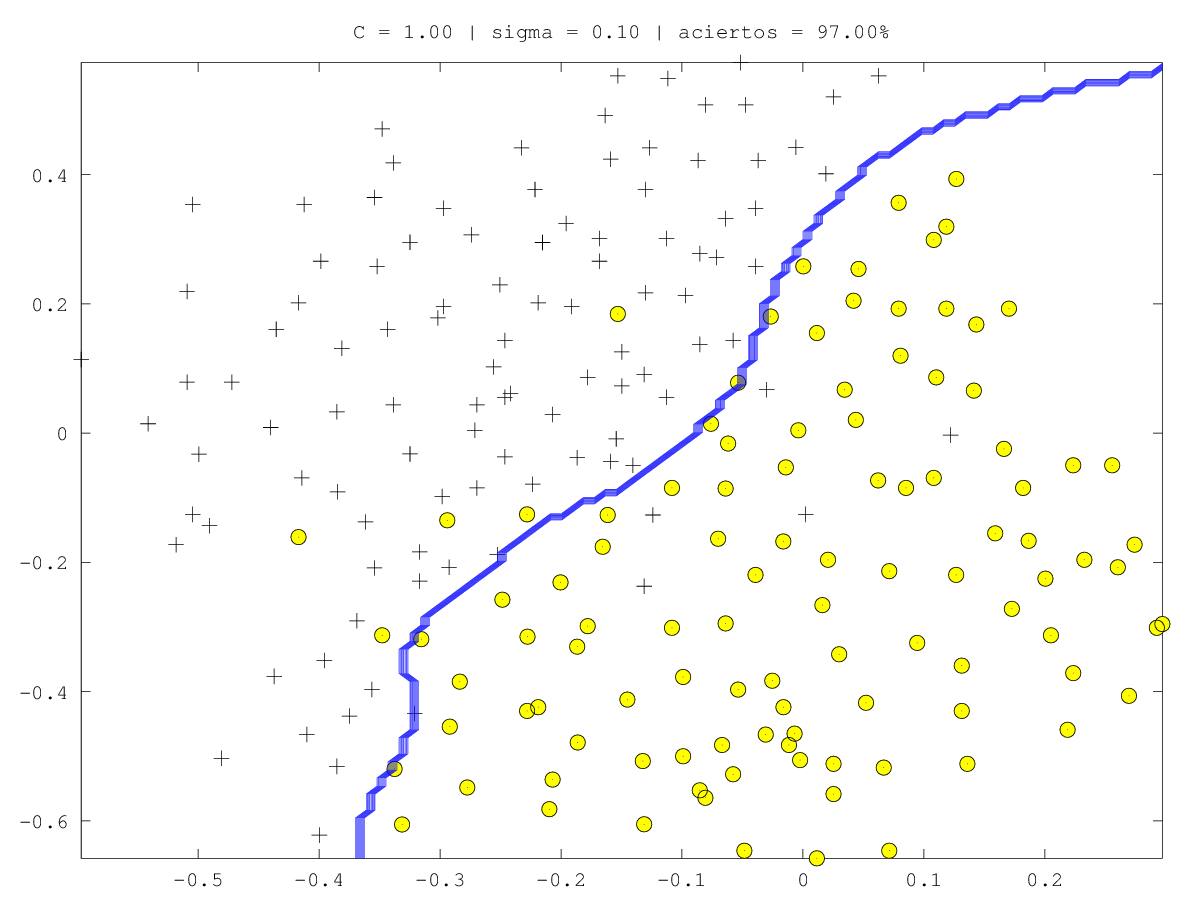
j = j+1;

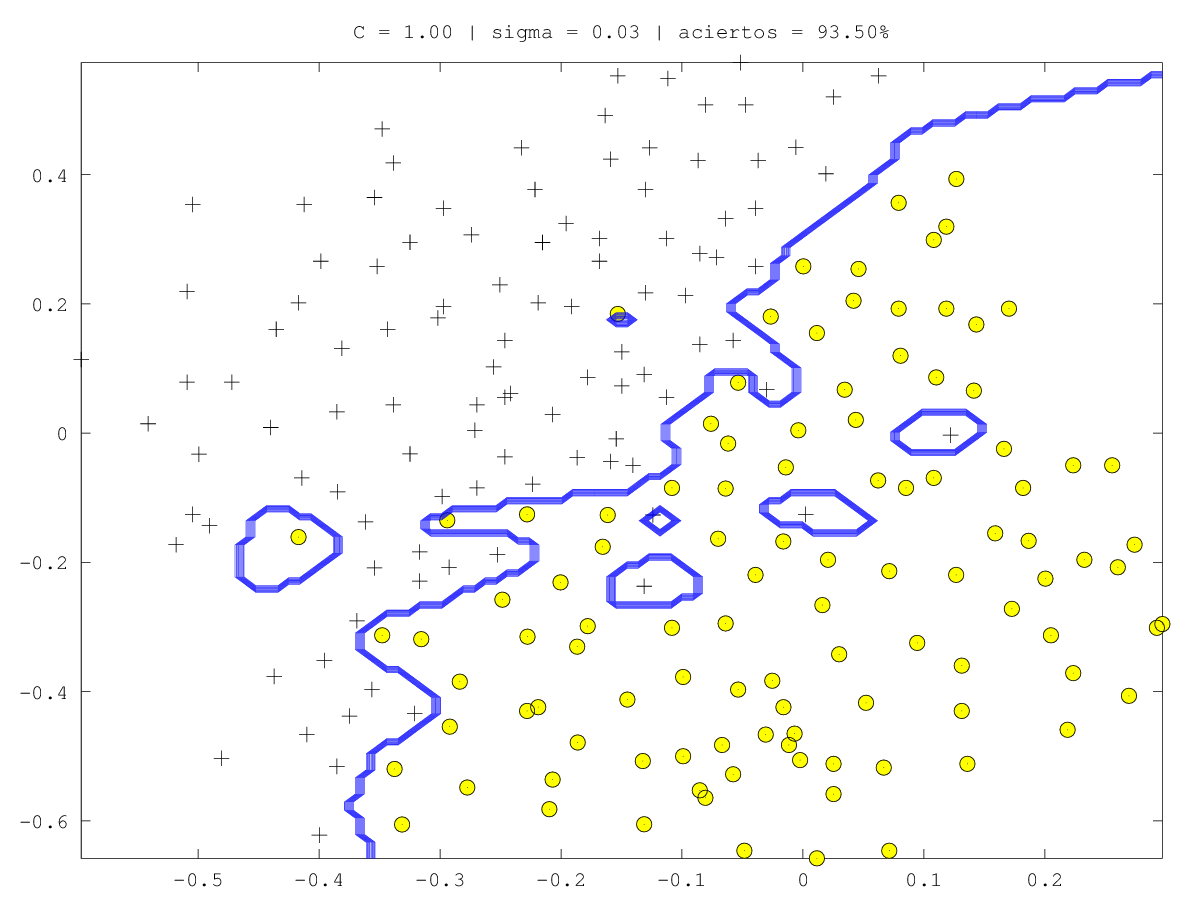
endfor

i = i+1;

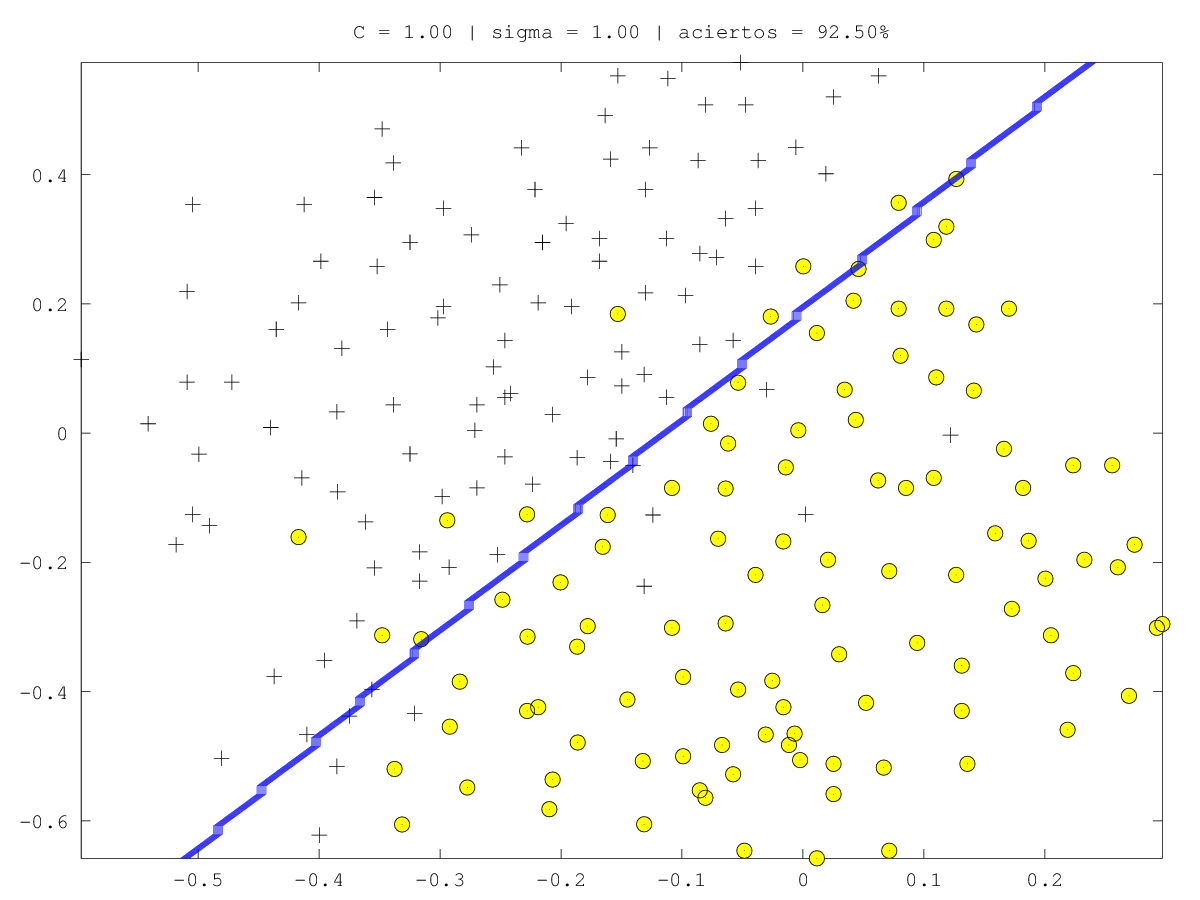
endfor

save dataSavePr666 dataSave;

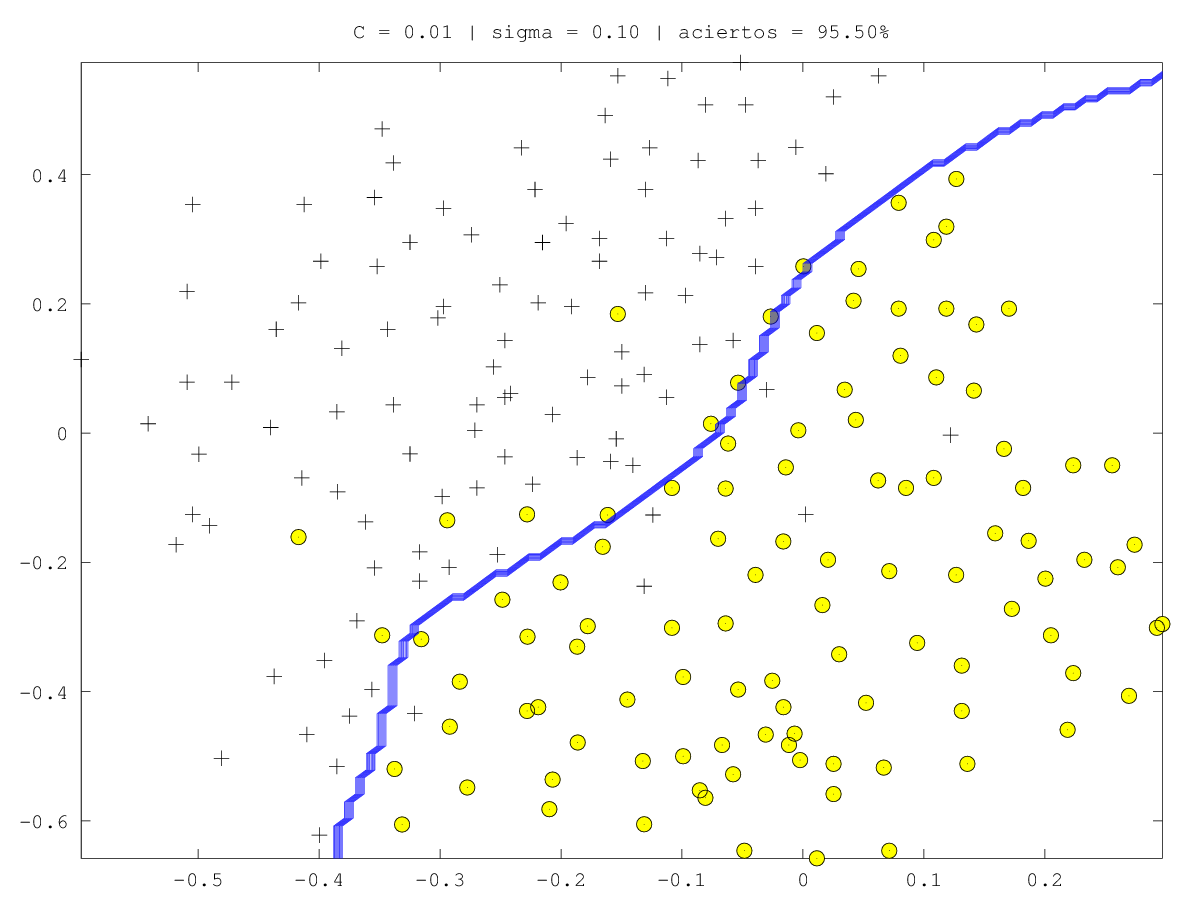


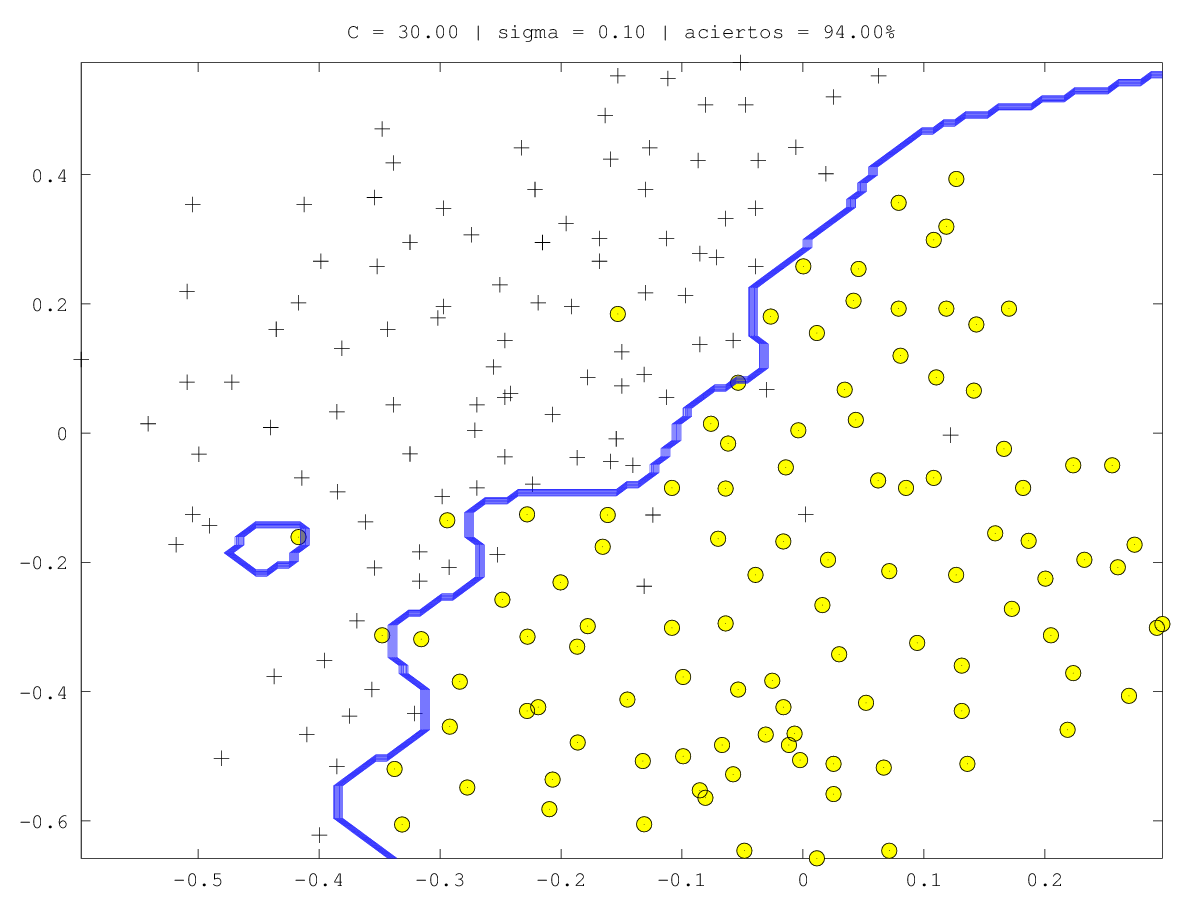


Apreciamos que, para el mismo valor de *C,* a menor valor de *sigma,* tiene a sobreajustarse.



En cambio, para un valor fijo de *sigma*, el sobreajuste aumenta con el mayor valor de *C.*





# filtro de spam

Lo primero es cargar el vocabulario en un formato comprensible y optimizado para Octave:

function vocabulario = cargaVocab()

vocablist = getVocabList;

for i = 1:length(vocablist)

vocabulario.(vocablist {i}) = i;

endfor

endfunction;

Lo siguiente es procesar los emails con el diccionario anterior. Como este proceso es bastante lento, conviene guardar los datos generados:

vocabulario = cargaVocab();

num\_easy\_ham = 2551;

num\_hard\_ham = 250;

num\_spam = 500;

for i = 1:num\_hard\_ham

nombreArchivo = sprintf('hard\_ham/%.4d.txt', i);

file\_contents = readFile(nombreArchivo);

email = processEmail(file\_contents);

X\_hard\_ham(i,:) = getX(email, vocabulario,tamVocabulario);

if (mod(i, 25) == 0) % Guardar datos intermedios

fprintf("Saving X\_hard\_ham\nIteración: %d\n", i);

save X\_hard\_ham X\_hard\_ham;

endif

endfor

save X\_hard\_ham X\_hard\_ham;

(…)

Se procede de manera análoga a los otros tipos de email

function x = getX(email, vocab, tamVocabulario)

x = zeros(1, tamVocabulario);

while ~isempty(email)

[str, email] = strtok(email, [' ']);

% procesa str

if (isfield(vocab, str))

x(getfield(vocab, str)) = 1;

endif

end

endfunction

El siguiente código es el utilizado para entrenar la SVM.

## Carga de los datos y selección de parámetros

clear;

save\_inter\_data = true;

num\_iteraciones\_totales = 3;

num\_easy\_ham = 2551;

num\_hard\_ham = 250;

num\_spam = 500;

% Asignar división ejemplos de entrenamiento/validación/test

parte\_train = 0.8;

parte\_val = 0.2;

parte\_test = 0;

if (parte\_train + parte\_val + parte\_test > 1)

error("La suma de las divisiones de ejemplos de entrenamiento/validación/test\nes mayor que 1");

endif

if(parte\_train + parte\_val + parte\_test < 1)

warning("La suma de las divisiones de ejemplos de entrenamiento/validación/test\nes menor que 1, no se utilizarán todos los casos");

endif

% Asignar parte a tomar de easy/hard/spam del total de cada tipo

parte\_easy = 1;

parte\_hard = 1;

parte\_spam = 1;

if (parte\_easy > 1 || parte\_hard > 1 || parte\_spam > 1)

error("No puede haber una parte mayor que 1");

endif

if (parte\_easy < 1 || parte\_hard < 1 || parte\_spam < 1)

warning("No se están utilizando todos los casos de entrenamiento\nal no cojer todo el conjuto de spam/easy/hard");

endif

cargaDatos;

disp("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Entrenando SVM con %d permutaciones de los ejemplos de entrenamiento\ndivididos %.2f%% como entrenamiento y %.2f%% como validación\ndando un total de %d ejemplos de entrenamiento y %d de validacion\nconteniendo a su vez en %.2f%% de los easy\_ham, %.2f%% de los hard\_ham y %.2f%% de los spam.\nUtilizando los siguientes parámetros:\n", num\_iteraciones\_totales,parte\_train\*100, parte\_val\*100,num\_train, num\_val, parte\_easy\*100, parte\_hard\*100, parte\_spam\*100);

valoresC = [8]

valoressigma = [1.4]

### Codigo de carga datos

load X\_spam;

load X\_easy\_ham;

load X\_hard\_ham;

num\_easy\_ham = 2551;

num\_hard\_ham = 250;

num\_spam = 500;

% Mezclar (se explica más adelante)

X\_easy\_ham = X\_easy\_ham(randperm(size(X\_easy\_ham)(1)),:);

X\_hard\_ham = X\_hard\_ham(randperm(size(X\_hard\_ham)(1)),:);

X\_spam = X\_spam(randperm(size(X\_spam)(1)),:);

% Seleccionar cuantos (importante después de mezclar)

num\_easy\_ham = floor(num\_easy\_ham \* parte\_easy);

num\_hard\_ham = floor(num\_hard\_ham \* parte\_hard);

num\_spam = floor(num\_spam \* parte\_spam);

% Seleccionar el numero deseado de cada

X\_spam = X\_spam(1:num\_spam, :);

X\_hard\_ham = X\_hard\_ham(1:num\_hard\_ham, :);

X\_easy\_ham = X\_easy\_ham(1:num\_easy\_ham, :);

getXY;

### CODIGO DE GETXY

train\_easy\_ham = floor(num\_easy\_ham \* parte\_train);

train\_hard\_ham = floor(num\_hard\_ham \* parte\_train);

train\_spam = floor(num\_spam \* parte\_train);

val\_easy\_ham = floor(num\_easy\_ham \* parte\_val);

val\_hard\_ham = floor(num\_hard\_ham \* parte\_val);

val\_spam = floor(num\_spam \* parte\_val);

% Separar entrenamiento/validacion de cada tipo %

separa;

% Unir en una X por training/validacion %

num\_train = train\_easy\_ham + train\_hard\_ham + train\_spam;

num\_val = val\_easy\_ham + val\_hard\_ham + val\_spam;

X\_train = [X\_tra\_eh; X\_tra\_hh; X\_tra\_spam];

% Para las y, marcamos como 0 (no spam) los easy y hard, y como 1 los spam

Y\_train = [zeros(1,train\_easy\_ham),zeros(1,train\_hard\_ham),ones(1,train\_spam)];

% Mezclamos otra vez para que no queden juntos los easy/hard/spam

perm = randperm(size(X\_train)(1));

X\_train = X\_train(perm, :);

Y\_train = Y\_train(perm)';

(…)

Para los ejemplos de validación es análogo

### Codigo de separa

X\_tra\_eh = X\_easy\_ham(1:train\_easy\_ham,:);

X\_val\_eh = X\_easy\_ham(train\_easy\_ham+1:train\_easy\_ham+val\_easy\_ham,:);

X\_tra\_hh = X\_hard\_ham(1:train\_hard\_ham,:);

X\_val\_hh = X\_hard\_ham(train\_hard\_ham+1:train\_hard\_ham+val\_hard\_ham,:);

X\_tra\_spam = X\_spam(1:train\_spam,:);

X\_val\_spam = X\_spam(train\_spam+1:train\_spam+val\_spam,:);

Los códigos anteriores se encargan de cargar los datos para el entrenamiento, juntándolos todos en las matrices de X\_train, Y\_train, X\_val e Y\_val, de tal forma que los ejemplos de entrenamiento y validación queden bien distribuidos: hay la misma proporción, indicada como constante, de spam en X\_train e X\_val, al igual que de easy\_ham y hard\_ham.

## parte de entrenamiento

for k = 1:num\_iteraciones\_totales

printf("Cargando los datos de la iteracion k = %d\n", k);

cargaDatos;

disp("Datos cargados!\n");

i = 1;

for C = valoresC

j = 1;

for sigma = valoressigma

printf("Iteracion k=%d, i=%d, j=%d\ncon C=%d sigma=%d\n",k,i,j,C,sigma);

model = svmTrain(X\_train, Y\_train, C, @(x1, x2) gaussianKernel(x1, x2, sigma));

ypred = svmPredict(model, X\_val);

if (save\_inter\_data)

dataSave(i,j).C = C;

dataSave(i,j).sigma = sigma;

dataSave(i,j).model = model;

% Porcentaje de aciertos

dataSave(i,j).porcentaje = size(find(Y\_val == ypred))(1) / size(Y\_val)(1) \* 100;

% Porcentaje, de los errores, que se predijo spam (y no lo era)

dataSave(i,j).porcentajePredErroneasSpam = size(find(ypred(find(ypred != Y\_val))==1))(1) / size(ypred(find(ypred!=Y\_val)))(1);

endif

j = j+1;

endfor

i = i+1;

endfor

endfor

Inicialmente, los datos de entrenamiento no se mezclaban y se obtuvieron los siguientes resultados:

Una vez entrenado con los valores de sigma y C

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

arrayPorcentajesVal =

C\s

22.475 22.475 22.657 66.606 55.869 70.792 71.429 28.025

22.475 22.475 22.657 66.697 57.507 70.246 71.793 29.299

22.475 22.475 22.657 66.697 57.052 69.882 71.702 27.934

22.475 22.475 22.657 66.697 57.052 69.518 71.429 28.571

22.475 22.475 22.657 66.697 56.506 75.705 81.984 28.753

22.475 22.475 22.657 66.697 58.417 80.801 83.167 29.026

22.475 22.475 22.657 66.697 58.417 82.257 82.348 30.482

22.475 22.475 22.657 66.697 58.417 82.257 85.441 28.662

Se observa que el maximo está en C30 sigma10, sin embargo para X\_test el porcentaje es de 65,5%

arrayPorcentajesTest =

C\s

17.951 17.951 20.036 45.422 38.259 56.392 51.859 20.036

17.951 17.951 20.036 45.422 38.622 55.848 54.216 19.674

17.951 17.951 20.036 45.422 38.622 55.576 54.125 19.130

17.951 17.951 20.036 45.422 38.622 55.848 53.762 19.402

17.951 17.951 20.036 45.422 37.625 62.647 63.191 19.583

17.951 17.951 20.036 45.422 38.622 66.818 62.194 19.946

17.951 17.951 20.036 45.422 38.622 68.359 62.919 20.218

17.951 17.951 20.036 45.422 38.622 68.359 65.549 18.858

prueba con C=linspace(3,30,8) sigma=(3,30,8)

arrayPorcentajesVal =

C\s

80.801 79.072 82.712 73.066 55.687 39.490 32.302 29.117

81.893 80.255 80.528 70.519 58.872 47.316 36.397 27.571

82.257 81.074 80.164 68.517 54.868 45.769 39.035 30.664

82.257 82.075 81.711 68.517 52.684 42.857 36.943 33.303

82.257 82.530 81.802 68.881 53.321 41.310 35.669 32.029

82.257 82.803 82.530 70.337 54.504 40.764 34.304 30.209

82.257 83.167 82.712 71.065 54.231 41.947 33.667 29.572

82.257 83.439 83.348 71.884 55.687 42.766 33.485 28.662

arrayPorcentajesTest =

C\s 3.000 6.8571 10.7143 14.5714 18.4286 22.2857 26.1429 30.0000

66.818 66.727 60.925 47.325 31.460 23.481 21.215 19.946

68.087 64.098 59.655 44.062 33.092 25.929 22.031 19.583

68.359 65.186 60.381 42.792 30.553 25.748 22.393 20.127

68.359 65.095 61.378 42.974 30.281 24.932 21.396 20.399

68.359 65.005 62.194 43.427 30.281 24.116 20.943 19.946

68.359 65.277 63.101 44.152 30.734 23.935 20.671 19.130

68.359 65.277 63.282 45.240 31.550 24.751 20.490 18.767

68.359 65.549 64.007 46.510 32.366 25.113 20.852 18.858

prueba con C=linspace(1,10,8) sigma=linspace(0.2, 10, 8);

arrayPorcentajesVal =

C= 1.0000 2.2857 3.5714 4.857 6.1429 7.4286 8.7143 10.0000

C\s0.2000 1.6000 3.0000 4.4000 5.8000 7.2000 8.6000 10.000

33.485 60.510 75.705 71.520 74.795 80.346 82.439 81.893

33.485 68.699 80.619 75.159 76.342 80.710 82.985 83.258

33.485 68.699 80.983 74.704 75.796 80.164 82.803 82.803

33.485 68.699 81.711 74.704 76.524 79.618 82.803 82.530

33.485 68.699 81.893 74.704 76.706 80.710 82.348 82.621

33.485 68.699 82.348 74.977 76.888 81.802 82.621 82.439

33.485 68.699 82.257 75.159 76.797 82.166 82.985 81.711

33.485 68.699 82.257 75.159 76.797 82.348 83.530 82.439

arrayPorcentajesTest =

C= 1.0000 2.2857 3.5714 4.857 6.1429 7.4286 8.7143 10.0000

C\s0.2000 1.6000 3.0000 4.4000 5.8000 7.2000 8.6000 10.000

33.001 42.067 62.647 63.554 63.554 65.186 65.911 63.101

33.001 48.232 67.543 66.364 65.911 66.727 64.642 63.373

33.001 48.232 66.818 66.727 64.551 66.092 65.095 62.103

33.001 48.232 67.180 66.546 64.098 64.642 65.095 62.375

33.001 48.232 67.724 64.551 64.461 64.370 64.914 62.375

33.001 48.232 68.359 64.733 64.642 64.007 64.642 61.922

33.001 48.232 68.359 64.823 64.551 64.642 64.551 62.829

33.001 48.232 68.359 64.914 64.551 65.549 64.733 62.919

Se observa una diferencia muy extraña entre los ejemplos de validación/testeo, ya que simplemente son ejemplos tomados en diferentes posiciones. Esto lleva a pensar que los ejemplos no están distribuidos de manera uniforme (de "dificultad de aprendizaje"), por lo que los mezclamos.

Viendo que al mezclar los elementos el porcentaje de validación/testeo coincide, se concluye que los datos no estaban bien distribuidos. Volvemos a probar valores para sigma y C, ahora sólo para X\_Val (dist 0.5Train 0.5Val 0Test)

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

arrayPorcentajesVal =

22.061 22.303 31.515 77.818 78.303 89.394 87.879 68.242

22.061 22.303 31.515 77.879 78.303 88.909 87.879 73.212

22.061 22.303 31.515 77.879 78.606 88.970 87.879 73.576

22.061 22.303 31.515 77.879 78.545 89.455 88.424 71.455

22.061 22.303 31.515 77.879 78.909 92.788 91.879 71.636

22.061 22.303 31.515 77.879 80.727 94.606 94.485 44.000

22.061 22.303 31.515 77.879 80.727 94.545 92.667 59.212

22.061 22.303 31.515 77.879 80.727 94.545 91.091 57.273

Se observa que el valor óptimo de sigma aparenta estar entre 1-10. El de C en torno a 3.

Probamos para sigma = linspace(1,10,8) C = linspace(1,4,8)

arrayPorcentajesVal =

86.788 93.758 91.455 90.727 90.182 88.970 87.939 86.788

88.909 94.606 93.091 92.000 91.697 91.091 90.182 88.606

90.000 94.848 93.818 92.485 91.818 91.455 91.273 89.697

90.000 94.848 93.818 92.545 91.818 91.515 91.697 90.424

90.000 94.848 93.818 92.485 91.636 91.576 91.455 90.485

90.000 94.848 93.818 92.485 91.455 91.152 91.091 90.424

90.000 94.848 93.697 92.424 91.333 90.909 91.030 90.242

90.000 94.909 93.636 92.485 91.394 90.848 90.727 90.061

Donde se ve que el valor sigma=1.4 es muy bueno.

Sin embargo, al ejecutar varias veces el enternamiento con distintas permutaciones de los ejemplos de entrenamiento/validacion para sigma= 1,4 y C=8, se observa que los porcentajes de acierto varían hasta un 10% :

arrayPorcentajesVal =

80.000

84.667

90.485

84.727

82.061

86.424

86.788

87.273

85.091

82.364

Dichos porcentajes se han calculado con un 50% ejemplos de entrenamiento/50% datos de valicadión

porcentajes\_50\_50 porcentajes\_30\_20 porcentajes\_50\_20 porcentajes\_80\_20

80.000 81.818 87.273 86.970

84.667 82.424 81.515 86.818

90.485 81.364 85.606 91.212

84.727 84.545 89.242 88.485

82.061 86.364 88.030 90.303

86.424 92.424 85.303 90.303

86.788 84.394 84.545 88.636

87.273 77.121 86.212 88.485

85.091 79.545 84.242 90.455

82.364 84.545 86.061 90.455

Avg 83.455 avg 85.803 avg 89.212

std 4.1591 std 2.1570 std 1.5498

Se observa que para un número fijo de número de elementos de validación, aumentar el número de ejemplos de entrenamiento ayuda a aumentar el porcentaje de aciertos sobre los ejemplos de validación, y lo más importante, la salida depede menos de que ejemplos se han seleccioando.

Porcentaje, en las predicciones erróneas, en las que se predijo spam:

> for i=1:k

> model = datosk(i).model;

> ypred = svmPredict(model, X\_val);

> size(find(ypred(find(ypred != Y\_val))==1))(1) / size(ypred(find(ypred!=Y\_val)))(1)

> endfor

ans = 0.85714

ans = 1

ans = 0.93750

ans = 1

ans = 0.93750

ans = 1

ans = 1

ans = 0.90909

ans = 0.92308

ans = 1

De lo que se extrae que el sistema clasifica muy bien el no spam como no spam, pero un poco peor al clasificar spam como no spam. Al cambiar los porcentajes de easy\_ham, hard\_ham y spam practicamente sólo se consigue empeorar el resultado, lo que puede indicar simplemente falta de ejemplos.

Probando para valores de C con sigma fijo a 1.14

Porcentaje Aciertos

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | 0.01000 | 0.10000 | 0.30000 | 1.00000 | 3.00000 | 8.0000 | 5.00000 | 10.00000 | 50.0000 |
| Avg | 82.93939 | 83.15152 | 83.45455 | 85.72727 | 88.93939 | 89.0000 | 89.21212 | 89.30303 | 87.8788 |
| Std | 2.33893 | 2.32614 | 2.37980 | 2.47207 | 2.24989 | 2.1868 | 2.23041 | 2.26413 | 1.6424 |

Con lo que se muestra que C~8 es un buen valor

## Conclusiones

Los datos inicialmente no estaban bien distribuidos, ya que algunos eran más difíciles de clasificar que otros. Para arreglar esto, simplemente se mezclan.

Que no sean todos iguales de fáciles de clasificar implica que según la permutación que se coja puede dar un porcentaje de aciertos mayor o menor, por lo que para hacer que el porcentaje de aciertos varíe menos se toman más ejemplos de entrenamiento.

Cuando falla, suele hacerlo clasificando como spam un correo que no era. Cambiar el porcentaje de ejemplos easy\_ham, hard\_ham o spam usado no cambiaba este resultado, por lo que esto puede significar simplemente falta de ejemplos.