ECE/CS 559 Neural Networks, Fall 2017 - Homework #8
Due: 11/17/2017, the end of class.

Erdem Koyuncu

All the notes in the beginning of Homework #1 apply.

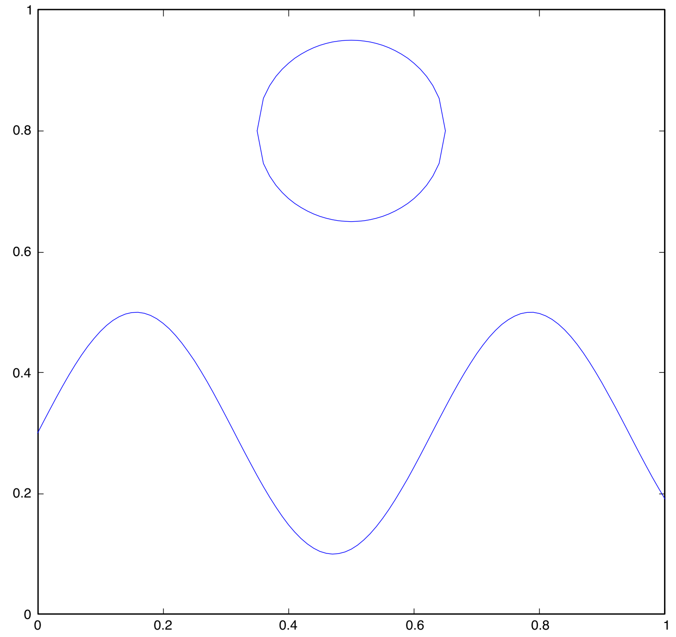
1. **(100pts)** In this computer project, we will design an SVM. You may use an existing library for solving the quadratic optimization problem that is associated with the SVM. For example, the free software Octave has such a command named "quadprog" to solve such problems. Other than that, you cannot use any existing machine learning/SVM library. As usual, please include the computer codes in your report.

   (a) Draw 100 points $\mathbf{x}_1, \ldots, \mathbf{x}_{100}$ independently and uniformly at random on $[0,1]^2$. These will be our input patterns.

   (b) Given $\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$, let

$$d_i = \begin{cases} 1, & x_{i2} < \frac{1}{5}\sin(10x_{i1}) + 0.3 \text{ or } (x_{i2} - 0.8)^2 + (x_{i1} - 0.5)^2 < 0.15^2 \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

   denote the desired classes corresponding to the input patterns. Let $C_1 = \{\mathbf{x}_i : d_i = 1\}$ and $C_{-1} = \{\mathbf{x}_i : d_i = -1\}$. The region where d_i is 1 is the union of the region that remains below the mountains and the region that remains inside the sun in the figure below. Sketch the points $\mathbf{x}_i$, indicate (with different colors or markers) their desired class. **Make sure your samples include at least one point inside the sun.**



   (c) Pick an appropriate kernel (write it down in your report) and design an SVM that separates the classes C_1 and C_{-1}. Recall that the result of the SVM will be a discriminant function $g(\mathbf{x}) = \sum_{i=1}^{\mathcal{I}_s} \alpha_i d_i K(\mathbf{x}_i, \mathbf{x}) + \theta$, where α_i are some positive constants, $\mathcal{I}_s$ is the set of the indices of support vectors, and θ is the optimal bias. Provide a rough sketch of the decision boundaries

$$\mathcal{H} \triangleq \{\mathbf{x} : g(\mathbf{x}) = 0\}$$
$$\mathcal{H}^+ \triangleq \{\mathbf{x} : g(\mathbf{x}) = 1\}$$
$$\mathcal{H}^- \triangleq \{\mathbf{x} : g(\mathbf{x}) = -1\}.$$

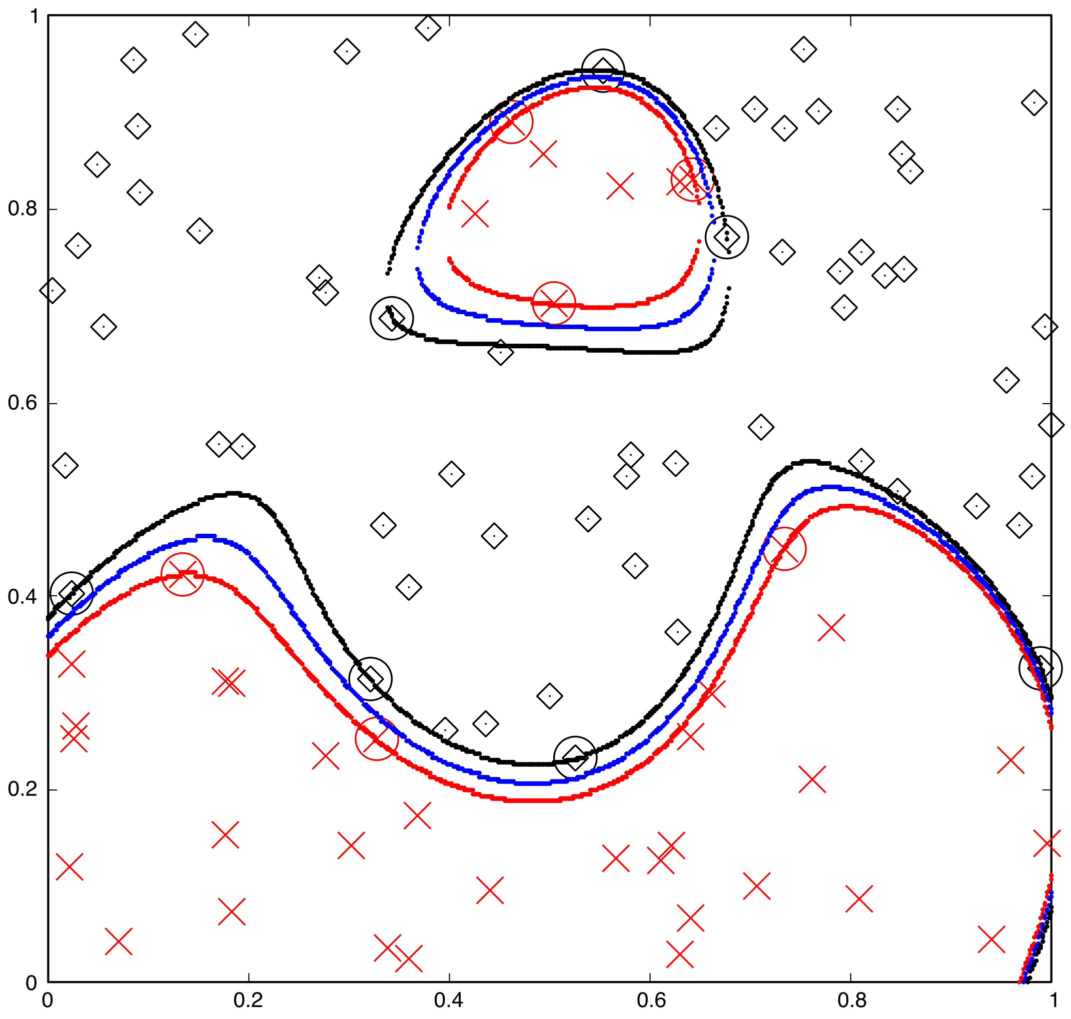
together with the support vectors. Note that the support vectors will be either on $\mathcal{H}^+$ or $\mathcal{H}^-$. **Your SVM should be able to perfectly separate the two classes with no exceptions of misclassified input patterns.**

Example solution: What I am expecting is a figure like this. The red crosses are input patterns with desired class 1, and the black diamonds are the input patterns with desired class -1. The decision boundary $\mathcal{H}$ (the blue line) can perfectly separate the two classes that were otherwise not linearly separable. There are 7 support vectors (marked by black circles) on $\mathcal{H}^+$ (the black line), and 6 support vectors (marked by red circles) in $\mathcal{H}^-$ (the red line) for a total of 13 support vectors. As expected, there are no patterns in between the "guard lines" $\mathcal{H}^+$ and $\mathcal{H}^-$. Of course, the Kernel that I used is a secret.

Note that plotting the decision boundary $\mathcal{H} \triangleq \{\mathbf{x} : g(\mathbf{x}) = 0\}$ (or plotting $\mathcal{H}^+$ or $\mathcal{H}^-$) as a nice perfect line is quite a difficult task. What you can do instead is to approximate it as:

```
for $x_1 = 0, 0.001, 0.002, \ldots, 0.999, 1$
    for $x_2 = 0, 0.001, 0.002, \ldots, 0.999, 1$
        if $g(\mathbf{x})$ is close enough to 0, then put a point in coordinates $(x_1, x_2)$.
    end
end
```

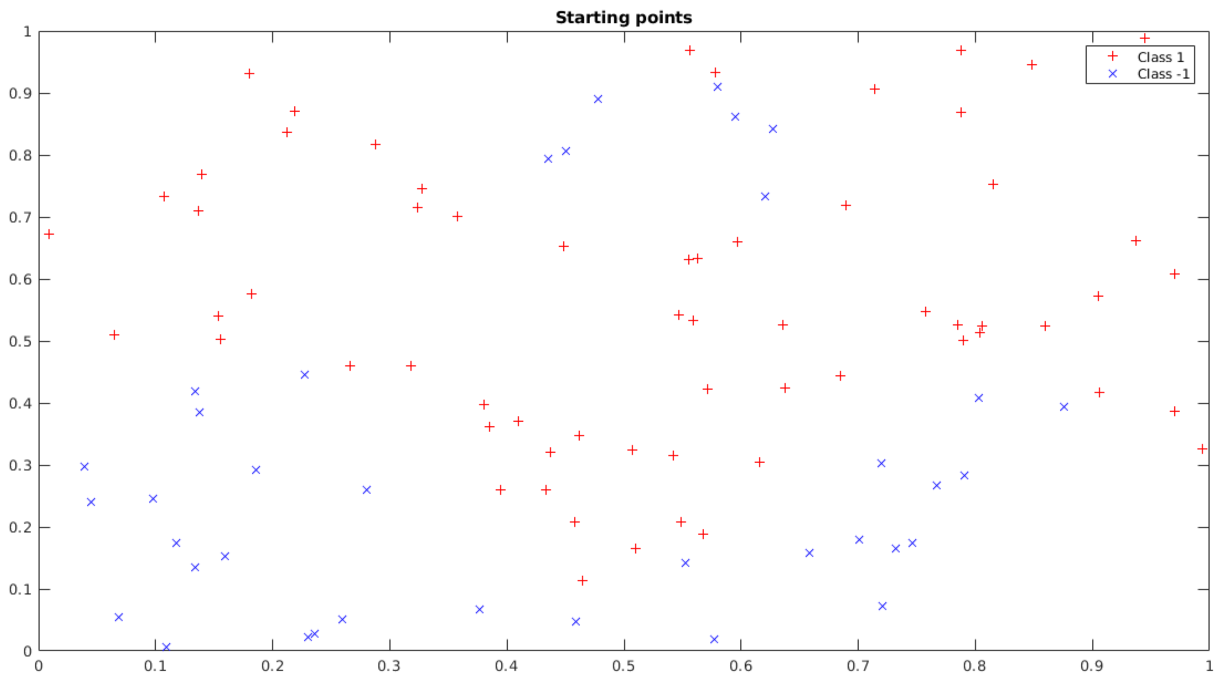
That is more-or-less what I did in my own figure (That is why the decision boundaries are not perfect and there are some "gaps" in between, you can even see the points that form the lines when you zoom in the PDF). Of course, you can use better techniques for generating the boundary curves if you like.



Olivo Iacopo – Homework 8

In order to solve the problem a matlab script has been developed.
The problem can be divided in three fundamental points:

- Ensure at least one input is inside the “sun”:
  In order to march this requirement a loop has been implemented when creating the random elements and plotting them. If no elements are inside the sun, the whole set is discarded and a new one is generated.
  Hence the resulting input set is:

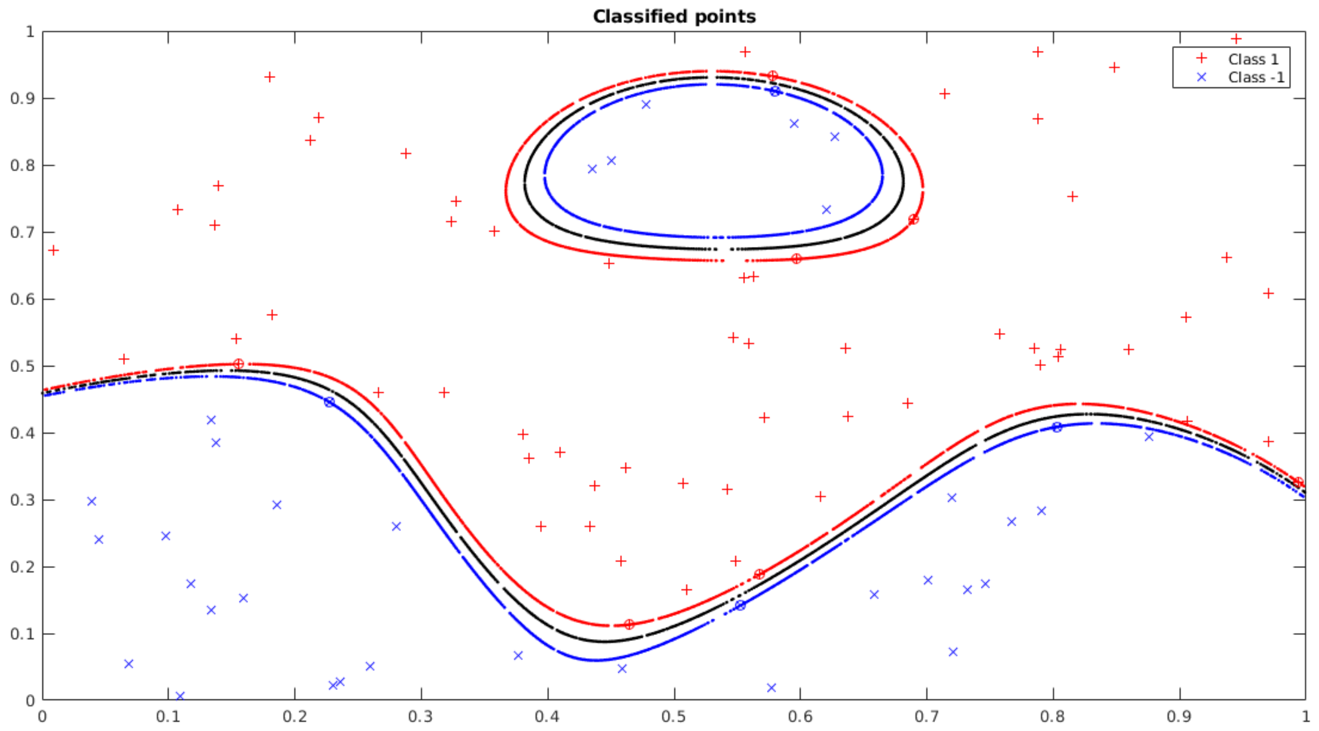


- Solve the MAX problem:
  In order to calculate all the lagrangian coefficients α, we need to solve a MAX problem. This can be done in mathlab through the function
  quadprog(H, f, A, b, Aeq, beq)
  Which returns the array of coefficients which maximize the function $\min_x : \frac{1}{2} x^T H x + f^T x$ with constraints $A x \leq b$ and $Aeq\, x = beq$. Our problem, however, has a different shape:
  $$MAX_{\alpha_i} : \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j K(x_i, x_j) \text{ with constraints } \alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i d_i = 0 \ .$$
  We need to reshape our formula in order to match the one solved by matlab. In order to do so:
  - we change the problem from a MAX to a min, since $MAX_x f(x) = \min_x - f(x)$
  - $$H = \begin{bmatrix} d_1 d_1 K(x_1, x_1) & \cdots & d_1 d_n K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ d_n d_1 K(x_n, x_1) & \cdots & d_n d_n K(x_n, x_n) \end{bmatrix}$$
  - $Aeq = [d_1, \cdots, d_n]$

    - $beq=0$
    - $f=\begin{bmatrix}-1\\ \vdots\\ -1\end{bmatrix}$
    - $b=\begin{bmatrix}0\\ \vdots\\ 0\end{bmatrix}$
    - $A=-I$
- Choose the kernel function:
  Since the two sets are not linearly separable, the best approach is to apply a kernel function in order to transform the input space to a space where we can separate the two classes by means of an hyperplane. We studied two kind of kernels:
    - Gaussian kernel function
    - Polynomial kernel function

  The Gaussian kernel has been chosen over the polynomial one due to the shape of the function and over the average better result it provides.
  It is fundamental to use a kernel function, since the Gaussian kernel maps the input space to an infinite-dimensional space where $\phi(x)^T\phi(x)$ is impossible to calculate.

After applying all the functions and calculating $g(x)=\sum_{i=1}^{n}\alpha_i b_i K(x_i,x)+\Theta$ we can try to classify all the input data, obtaining:



We can clearly note how the input classes are separated by means of the calculated function. In the plot it is possible to note the decision boundaries $H^{+¿}$, H and $H^{-¿}$.
All the inputs result in being correctly classified. By paying more attention to the plot, we can note 11 support vectors: 7 lying on $H^{-¿}$ and 4 lying on $H^{+¿}$.

```
close all
clear all
clc

n = 100;
precision = 10000;

isvalid = false;
D = zeros(n,1);

%% Data creation
while ~isvalid
    X = rand(2,n);

    % Plot the graph
    figure(1)
    for i = 1:1:n
        D(i) = di(X(:,i));
        if D(i) > 0
            plot(X(1,i), X(2,i), 'xb'), hold on
            % Check at least one element is inside the sun
            if (X(2,i) - 0.8)^2 + (x(1,i) - 0.5)^2 < 0.15^2
                isvalid = true;
            end
        else
            plot(X(1,i), X(2,i), '+r'), hold on
        end
    end
end


%% Problem
% Get all the alpha
K = zeros(n);
for i = 1:1:n
    for j = 1:1:n
        K(i,j) = kernel(X(:,i), X(:,j));
    end
end
H = (D.*D').*K;
b = zeros(n,1);
A = eye(n) .* (-1);
beq = 0;
Aeq = D';
f = ones(n,1) .* (-1);

a = quadprog(H,f,A,b,Aeq,beq);
a = round(a,4);
% Calculate W
W = sum(a'.*D'.*X,2);

% Calculate all the support vectors
supports = find(a);
Xs = X(:,supports);
Ds = D(supports);
As = a(supports);
% Calculate teta


T = zeros(size(supports,1), 1);
for i = 1:1:size(supports)
    T(i) = Ds(i) - sum(a.*D.*K(:,supports(i)));
```

```
end
T = round(T,2);
for i = 1:1:size(supports)-1
    if T(i) ~= T(i+1)
       ERROR
    end
end



%% Plot the graph
figure(2)
for i = 1:1:n
    res = sign(sum(a.*D.*K(:,i)) + T(1));
    if res > 0
        plot(X(1,i), X(2,i), 'xb'), hold on
    else
        plot(X(1,i), X(2,i), '+r'), hold on
    end
end

%% Plot G(x)
for i = 0:1/precision:1
    for j = 0:1/precision:1
        K = zeros(size(supports,1),1);
        for k = 1:1:size(supports)
            K(k) = kernel(Xs(:,k), [i;j]);
        end
        g = round(sum(As.*Ds.*K) + T(1), 3);
        if g == 1
            plot(i,j,'.b'), hold on
        end
        if g == -1
            plot(i,j,'.r'), hold on
        end
        if g == 0
            plot(i,j,'.k'), hold on
        end
    end
end

%% Plot the support vectors
for i = 1:1:size(supports)
    if Ds(i) > 0
        plot(Xs(1,i), Xs(2,i), 'ob'), hold on
    else
        plot(Xs(1,i), Xs(2,i), 'or'), hold on
    end
end
```

```
function ret = di(x)
    ret = -1;
    if sin(10*x(1))/5 + 0.3 > x(2)
        ret = 1;
    end
    if (x(2) - 0.8)^2 + (x(1) - 0.5)^2 < 0.15^2
        ret = 1;
    end
end
```

```
function ret = kernel(xi, xj)
    % Gaussian kernel
    sigma = sqrt(1/2);
    ret = exp( - (norm(xi - xj)^2) / (2 * (sigma^2)));
end
```