Lab Assignment 5

Steven Truong

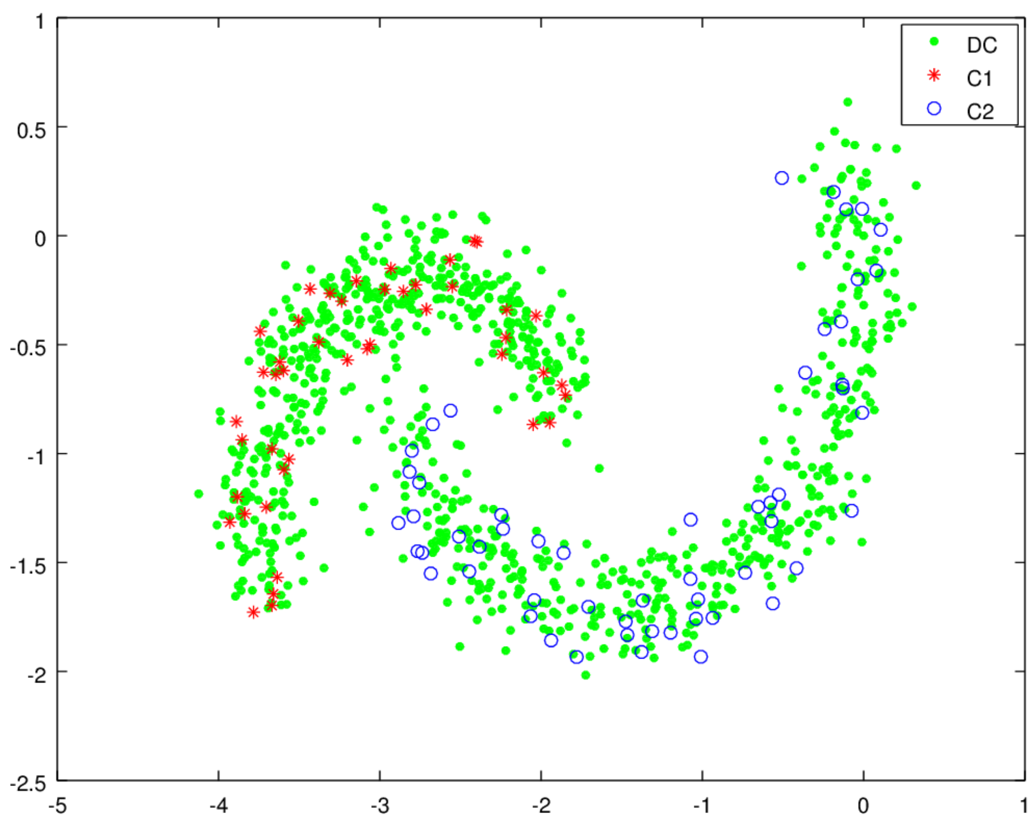
Task 1

The role of the term $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$ is to penalize large values of $\boldsymbol{w}$. Given $\boldsymbol{w}$ which classifies the data properly, $\alpha\boldsymbol{w}$ will do the same, which means that we can have arbitrarily large values of $\boldsymbol{w}$, which is unfavorable, since large values of $\boldsymbol{w}$ give an unstable classifier.

Task 2

If $\boldsymbol{x}_n$ is close to $\boldsymbol{x}_m$, we would want to put them in the same class. So, we want to make $k_n(\boldsymbol{x}) = K(\boldsymbol{x}_m, \boldsymbol{x}_n)$ large so that t_n has the most weight in formula (2), which would more likely give us the correct classification.

Task 3



Task 4

No, I don't think there is a simple transformation φ which would transform the data set into something linearly separable. The data looks like a swirl, so we'd need to warp it into two "lines" by some kind of rotation-like transformation, which would not be easy.

Task 5

$$
\begin{aligned}
\nabla_{\boldsymbol{w}} J(\boldsymbol{w}, \lambda) = \sum_{\boldsymbol{x}_n \in \mathcal{C}} (\varphi(\boldsymbol{x}_n)^\top \boldsymbol{w} - t_n)\varphi(\boldsymbol{x_n}) + \lambda \boldsymbol{w} = 0 \\
\iff \sum_{\boldsymbol{x}_n \in \mathcal{C}} \varphi(\boldsymbol{x}_n)\varphi(\boldsymbol{x}_n)^\top \boldsymbol{w} + \lambda \boldsymbol{w} = \sum_{\boldsymbol{x}_n \in \mathcal{C}} t_n \varphi(\boldsymbol{x}_n) \\
\iff \left(\lambda I_m + \sum_{\boldsymbol{x}_n \in \mathcal{C}} \varphi(\boldsymbol{x}_n)\varphi(\boldsymbol{x}_n)^\top\right)\boldsymbol{w} = \sum_{\boldsymbol{x}_n \in \mathcal{C}} t_n \varphi(\boldsymbol{x}_n).
\end{aligned}
$$

If we define

$$
\Phi := \begin{pmatrix} | & & | \\ \varphi(\boldsymbol{x}_1) & \cdots & \varphi(\boldsymbol{x}_n) \\ | & & | \end{pmatrix}
$$

$$
\boldsymbol{t} := \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix},
$$

then we can write the equation as $\big(\Phi\Phi^\top + \lambda I_m\big)\boldsymbol{w} = \Phi \boldsymbol{t}$.

```matlab
% phi: R^d -> R^d+1
function y = phi(x)
  y = [x; 1];
end

function w = findCoefficient(C, t, lambda)
  % Each column of C should be an observation
  N = size(t);

  % Calculate coefficients
  % n-th column is phi(x_n)
  Phi = phi(C(:, 1));
  for i = 2:N
    Phi = [Phi, phi(C(:, i))];
  end

  % Solve least squares (Phi * Phi + lambda * I)^T * w = Phi * t
  w = ((Phi * t) \ (Phi * Phi' + lambda * eye(size(C, 1) + 1)))';
end
```

Task 6

For very large λ ($\sim 10^7$), the classifier maxes out with around a 73.22% accuracy.

Task 8

```
load two_moons.mat

[dim, dataSize] = size(data);

global N C t k lambda;
N = 100;              % Number of classes considered known
C = data(:, 1:N);     % Each column is an observation
t = -2 * labels .+ 3; % Map 1 -> 1 and 2 -> -1

k = 8;        % k-th nearest neighbor to use
lambda = 0.1; % Arbitrary positive scalar

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Helper Functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculates the k-th nearest neighbor of x given the known data
function [s, y] = kthNN(x)
  global N C t k;
  [classified, k, dist, index] = fastKNN([C', t(1:N)], x', k);

  s = dist(k);
  y = C(:, index(k));

  % This happens if x is in C
  if (dist(1) == 0)
    s = dist(k + 1);
    y = C(:, index(k + 1));
  end
end

% Calculates K(x, y)
%   sx and sy are scaling factors
function y = kernel(x, y, sx, sy)
  y = exp(-norm(x - y)^2 / (sx * sy));
end

% Gives classification for x
%   sx is the scaling factor for x
%   Kinv is the inverse of K + lambda * I
%   s contains scaling factors for everything
%   t contains the classes of the data
function y = kernelClassify(x, sx, Kinv, s, t)
  global N C;
  for i = 1:N
    k(i) = kernel(x, C(:, i), sx, s(i));
  end
  y = k * Kinv * t;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Main Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Calculate sigmas for data
for i = 1:dataSize
  s(i) = kthNN(data(:, i));
end

% Calculate kernel matrix
for n = 1:N
  for m = 1:N
    K(n, m) = kernel(C(:, n), C(:, m), s(n), s(m));
  end
end

% Calculate inverse of a matrix K + lambda * I
Kinv = inverse(K + eye(N) * lambda);

% Calculate accuracy
accurate = 0;
for i = (N+1):dataSize
  if (sign(kernelClassify(data(:, i), s(i), Kinv, s, t(1:N))) == t(i))
    accurate = accurate + 1;
  end
end

accurate / (dataSize - N)
```

Task 9

Taking $\lambda = 0.1$ and $k = 8$, classification using the kernel method gives an accuracy of about 95.44%, which is significantly better than that of the least squares classifier.