Pattern Recognition Assignment 1

James Jackson

November 2015

Question 1

Part A

```
close all
clear all
clc

% 1 A
N = 10; % used throughout the script
% Generate distinct random points
r = randperm(25);
[x y] = ind2sub([5,5],r(1:N));
z = [x' y'];
figure
hold on
% have to increase marker size as it is
plot(z(:,1), z(:,2), 'k.', 'markersize', 30);
text(z(:,1) + 0.1, z(:,2) + 0.1, num2cell(1:N));
```

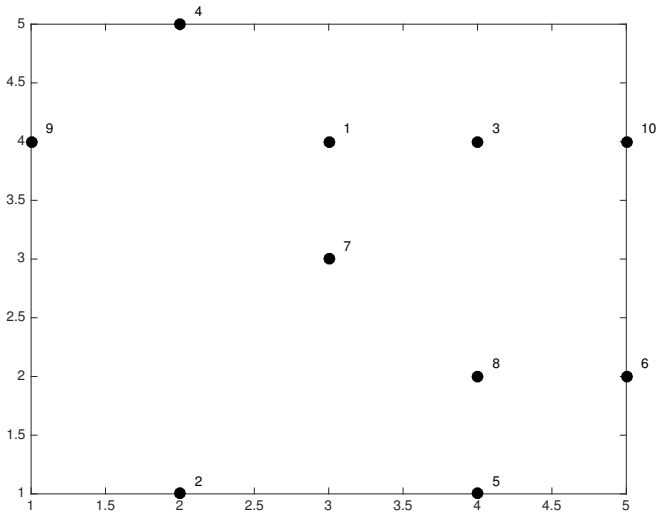

Figure 1: Random plots generated by Question 1 part A

Part B

```
% use repmat to create a matrix which contains the objects
% repeated in order
% use repelem to create a matrix containing the objects repeated
% in groups
% calculate distances between the matrices
% reshape the resulting vector to form an NxN matrix
d=reshape(sum((repmat(z,N,1)'-repelem(z,N,1)').^2),N,N);
```

Output

```
     0    10     1     2    10     8     1     5     4     4
    10     0    13    16     4    10     5     5    10    18
     1    13     0     5     9     5     2     4     9     1
     2    16     5     0    20    18     5    13     2    10
    10     4     9    20     0     2     5     1    18    10
     8    10     5    18     2     0     5     1    20     4
     1     5     2     5     5     5     0     2     5     5
     5     5     4    13     1     1     2     0    13     5
     4    10     9     2    18    20     5    13     0    16
     4    18     1    10    10     4     5     5    16     0
```

Part C

```
% Apply kruscals minimum spanning tree to the dataset and plot
% the clusters on the figure as coloured rings around the data
% values
a = kruskals_mst_with_comments(z, 3);
plot(z(a==1,1), z(a==1,2), 'ro', 'markersize', 10, 'linewidth', 10);
plot(z(a==2,1), z(a==2,2), 'go', 'markersize', 10, 'linewidth', 10);
plot(z(a==3,1), z(a==3,2), 'bo', 'markersize', 10, 'linewidth', 10);
```

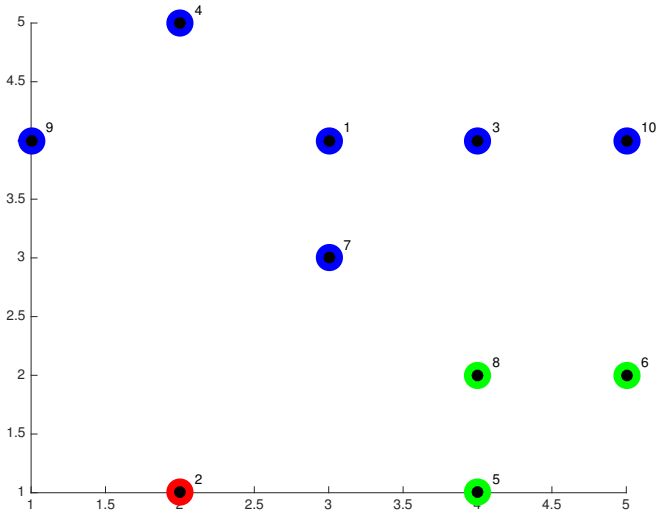

Figure 2: Clustered points

Part D

Kruskals Minimum Spanning Tree with Step Return

```
function [a, steps] = kruskals_mst_with_steps(b,c)
d = size(b,1);
e = nchoosek(1:d,2);
f = b(e(:,1),:) - b(e(:,2),:);
[~,g] = sort(sum(f.*f,2));
e = e(g,:);
% sort weights
f = sort(sum(f.*f,2));
% set the size of steps we will use the d+1 column to store the jump
steps = zeros(d, d+1);
a = 1:d;
i = 1;
% J will hold the jump length
J = 0;
p = 0;
while numel(unique(a)) > c
 if a(e(1,1)) ~= a(e(1,2))
      steps(i, :) = [a J];
      a(a==a(e(1,1))) = a(e(1,2));
      J = sqrt(f(1,:).^2);
      i = i+1;
      f(1,:) = []; % remove the shortest distance after use
 end


 e(1,:) = []; % remove the shortest edge after use
end
steps(i+1, :) = [a J]; % last step
steps(all(steps==0,2),:)=[]; % remove rows of zeros
a = cmunique(a) + 1;
```

Using the above function we now output the steps to the console

```
% get the clusters and the steps from the new function
[a, steps] = kruskals_mst_with_steps(z, 3);
[N, n] = size(steps);
index = 1 : n - 1; % minus 1 as the last colomn is the Jump
% loop over the steps returned
for i = 1 : N
   % get step array and Jump size
    [x J] = deal(steps(i,1:10), steps(i,11));
    u = unique(x);
    % use array fun to build a cell array of the values in the clusters
    line = arrayfun(@(a) sprintf('%s', sprintf('%d ', index(x==a))), ...
        u, 'uniformoutput', 0);
    % print the current step
    fprintf('Step %3i J = %3i, #Cl %3i: ( %s)\n',...
        i, J, numel(u) ,strjoin(line, ') ( '));
end
```

Output

```
Step   1 J =   0, #Cl  10: ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 )
Step   2 J =   1, #Cl   9: ( 2 ) ( 1 3 ) ( 4 ) ( 5 ) ( 6 ) ( 7 ) ( 8 ) ( 9 ) ( 10 )
Step   3 J =   1, #Cl   8: ( 2 ) ( 4 ) ( 5 ) ( 6 ) ( 1 3 7 ) ( 8 ) ( 9 ) ( 10 )
Step   4 J =   1, #Cl   7: ( 2 ) ( 4 ) ( 5 ) ( 6 ) ( 8 ) ( 9 ) ( 1 3 7 10 )
Step   5 J =   1, #Cl   6: ( 2 ) ( 4 ) ( 6 ) ( 5 8 ) ( 9 ) ( 1 3 7 10 )
Step   6 J =   1, #Cl   5: ( 2 ) ( 4 ) ( 5 6 8 ) ( 9 ) ( 1 3 7 10 )
Step   7 J =   2, #Cl   4: ( 2 ) ( 1 3 4 7 10 ) ( 5 6 8 ) ( 9 )
Step   8 J =   2, #Cl   3: ( 2 ) ( 5 6 8 ) ( 1 3 4 7 9 10 )
```

Part E

```
% get the step before the largest jump
% rerun function but get all steps back
[a, steps] = kruskals_mst_with_steps(z, 1)
% sort the differences between the critereon values (must assign a 0
% column to get the first difference of 0)
[d, i] = sort(diff( [0 steps(:,11)']));
% get the index of the largest jump with fewest clusters
S = i(end) - 1;

% get step array and Jump size
[x, J] = deal(steps(S,1:10), steps(S,11));
u = unique(x);
% use array fun to build a cell array of the values in the clusters
line = arrayfun(@(a) sprintf('%s', sprintf('%d ', index(x==a))), ...
    u, 'uniformoutput', 0);
% print the current step
fprintf('Suggested Clustering:\nStep %3i J = %3i, #Cl %3i: ( %s)\n',...
    S, J, numel(u) ,strjoin(line, ') ( '));
```

Output

```
Suggested Clustering
Step   6 J =   1, #Cl   5: ( 2 ) ( 4 ) ( 5 6 8 ) ( 9 ) ( 1 3 7 10 )
```

Question 2

Part A

```
% functions to generate data and labels
f = @() randn(400,2) * randi(6) - 3;
l = @(a) ones(400,1) + a;
Z = [f();f();f();f();f()];
labs = [l(-1); l(0); l(1); l(2); l(3)];
figure
hold on
% plot data ourlined
arrayfun(@(a) plot(Z(labs==a,1), Z(labs==a,2), '.', ...
    'color', rand(1,3), 'markersize', 20), 0:4)
```

Output

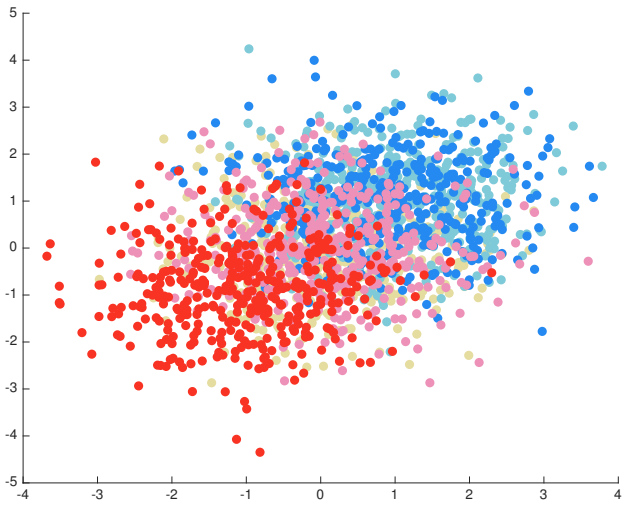

Figure 3: Data plotted from 2A

Part B

```
figure
for i = 2:7
    c = kmeans_cg(Z, i);
    subplot(2, 3, i-1), hold on, axis square off;
    % Draw subplots
    arrayfun(@(a) plot(Z(c==a,1), Z(c==a,2), '.', ...
    'color', rand(1,3), 'markersize', 20), 1:i)
    title(i);
end
```

Output

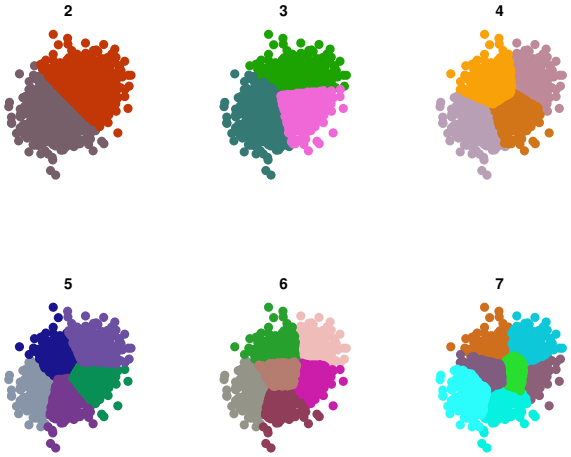

Figure 4: Clustered forms of data from 2a

Question 3

Part A

```
close all
clear all
clc

load('Example_MNIST_digits.mat');
D38 = b(labb==4 | labb==9, :);
% we are interested in 3 and 8 so minus one off labels to make for
% clearer code
l = labb(labb==4 | labb==9) - 1;
```

Part B

```
% we are doing 1 dimensional distance so only need the absolute
% distance between the means
d = abs(mean(D38(l==3,:))-mean(D38(l==8,:)));
% sort in descending order
[J, i] = sort(d,'descend');
fprintf('Feature\t\tJ\n===\t\t===\n');
fprintf('%i\t\t%.2f \n', [i(1: 10)' J(1: 10)']' );
```

Output

```
Feature    J
===        ===
488        149.10
489        148.48
462        141.63
461        139.66
516        138.49
515        130.65
490        122.05
463        119.22
517        117.49
487        117.24
```

Part C

```
figure, hold on, axis square off, colormap jet
imagesc(reshape(d,28,28));
```

Output

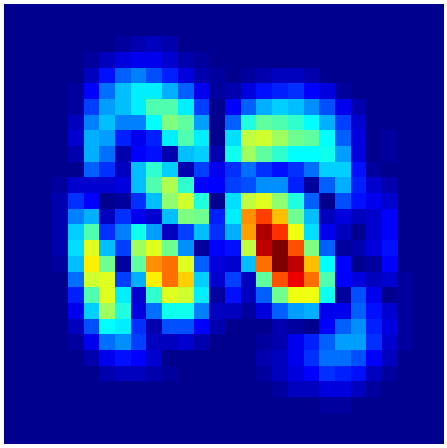

Figure 5: Image representation of criteron values

Part D

```
badF = sum(d==0)
figure
% flip the data set so that the 0's are at the begining
% this will allow for better visualisation of 0s
plot(fliplr(J), '-', 'linewidth', 5);
D38_clean = D38(:, d~=0);
```

Output

```
badF =
   235
```

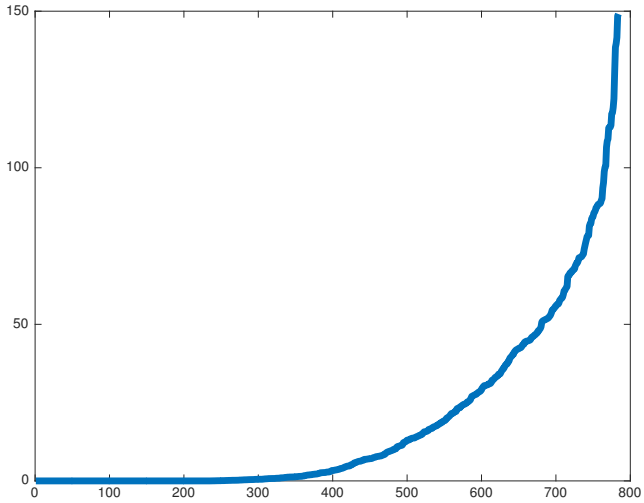

Figure 6: Plot of J values

Part E

```
[~,pc] = pca(D38_clean);
figure, hold on, axis square
plot(pc(l==3,1), pc(l==3,2), 'k.', 'markersize', 20);
plot(pc(l==8,1), pc(l==8,2), 'r.', 'markersize', 20);
legend('Digit 3', 'Digit 8');
```

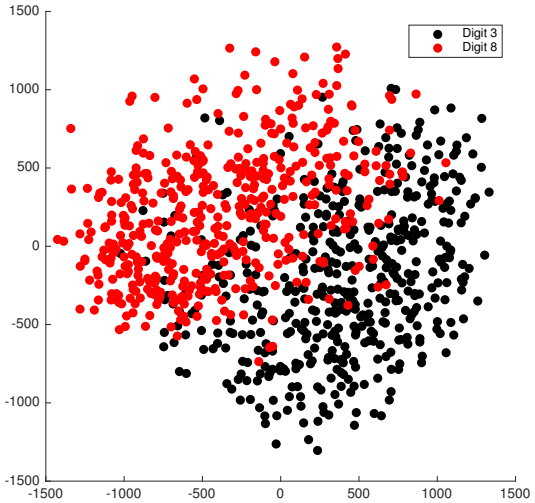

Figure 7: Plot of first two PC

Part F

```
% index for the top 8 features
ind = i(1: 8);
rE1 = 1 - mean(l == MyNMC(D38(:, ind), l, D38(:, ind)))
rE2 = 1 - mean(l == MyNMC(pc(:, 1:8), l, pc(:, 1:8)))
```

Output

```
rE1 =
  0.1421
rE2 =
  0.1285
```

The resubstitution errors between these two feature representations is very similar. In this case, selecting the top 8 features gives a marginally better result. Increasing the number of features selected continues the trend but closes the gap, for example. If we select the to 16 features and take the first 16 componentes our results are:

```
rE1 =
  0.1181
rE2 =
  0.1202
```

When we reduce the number to 4 there is still marginal difference between the two methods but PC performs better.

```
rE1 =
  0.1755
rE2 =
  0.1526
```

Principal component analysis analyses the variance between features and returns the highest to lowest in terms of this variance. For our 2D dataset the variance between features is the distance between them so our univariant feature selection is essentially applying the same metohd for ranking features which leads to the very similar outputs.

Question 4

```
function w = perceptron(d,l,epochs,E)
    m = [min(d(:,1)) max(d(:,1))];
    figure, hold on, axis square
    axis([m(1) m(2) min(d(:,2)) max(d(:,2))])
    u = unique(l);
    % Get sizes of dataset
    [p, q] = size(d);
    % initialise weights with small random numbers
    w = rand(q+1,1);
    errors = 1;
    count = 0;
    % infinite loop
    % while q*q'
    % could use rand(1,3) but similar colors make it hard to read
    c = [1 0 0; 0 1 0];
    while errors > 0 && count < epochs
        errors = 0;
        count = count + 1;
        % loop over objects
        for i = 1:p
            % get input for the perceptron including the bias
            z = [1 d(i,:)];
            % get output value which is the product of the inputs and
            % the weights less than 0 (will return 1 or 0 (T or F))
            v = z * w < 0;
            % q(i) will be 1 or 0 depending on whether v is the same as l(i)
            q(i) = v==l(i);
            errors = ~q(i);
```

```
            if errors
                % adjust weights according to the formula
                w = w + ( 2 * v - 1 ) * E * z';
                y1 = -(w(1) + w(2) * m(1)) / w(3);
                y2 = -(w(1) + w(2) * m(2)) / w(3);
                y = [y1 y2];
                plot(m,y,'k-');
            end

        end
    end
    % plot points
    % find y's based on max and min x's
    y1 = -(w(1) + w(2) * m(1)) / w(3);
    y2 = -(w(1) + w(2) * m(2)) / w(3);
    y = [y1 y2];
    % plot the line
    plot(m,y,'b-', 'linewidth', 3);
    % plot final line
    for i = 1:numel(u)
        plot(d(l==u(i),1), d(l==u(i), 2), '.', 'markersize', ...
            30, 'color', c(i,:));
    end
```

```
close all
clear all
clc
% create dataset
Z = randn(100,2);
% label according to random linear function
l = Z(:,1) > 0;

% Get the weights
w = perceptron(Z,l,0.1);
w = perceptron(Z,l,0.8);
```

Output

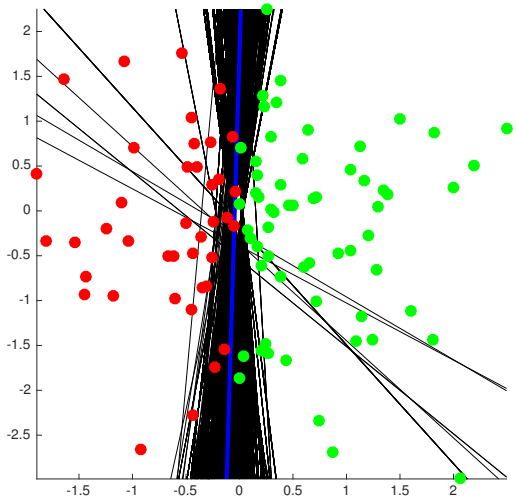

Figure 8: Perceptron Learning Rate 0.1

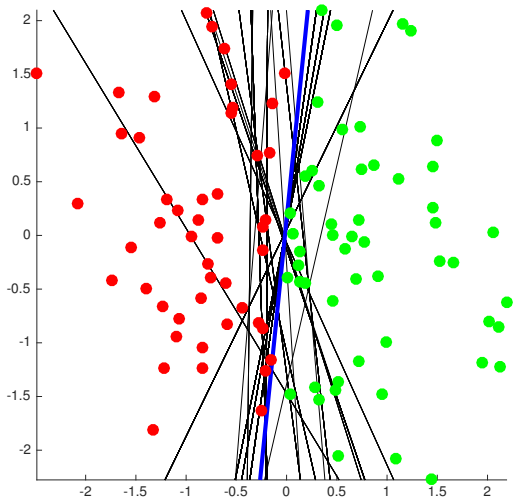

Figure 9: Perceptron Learning Rate 0.8

With a lower learning rate, the weights are updated in smaller increments. In the above case this has meant for much more iterations before converging, the heigher learning rate required fewer iterations. However a higher learning rate means a larger movement in the boundary so. This large movement could 'overshoot' the required boundary and lead to more iteratoins.

As a further test both learning rates were tested 100 times and the epochs counted giving the following means as a result.

```
lowLR =
   3.1100
highLR =
   4.9000
```

While far from exhaustive, this test implies that a higher learning rate is less efficient than a smaller learning rate from this data.

Question 5

```
close all
clear all
clc

% lazy copy and paste from pdf
d = [0.84 0.55 0.08 -0.64;
0.49 0.61 0.70 0.51;
0.67 0.25 0.59 1.06;
0.92 0.50 0.18 0.51;
0.94 0.66 0.30 -0.35;
0.22 0.85 0.70 -1.49;
0.21 0.78 0.84 1.53;
0.81 0.92 0.44 -0.50;
0.10 0.44 0.07 0.58;
0.21 0.70 0.41 1.89;]';

% assign more meaningful variables to work with
c = d(1:2,:)';
s = d(3,:)';
w = d(4,:)';
b = 0.28;
N = 1000;
[x, y] = meshgrid(linspace(0,1,N));
% get all points as x y columns
```

```
ps = [x(:) y(:)];
n = zeros(size(ps,1),1);
for p = 1: size(ps,1)
    % apply the rbf formula
    % repeat the current point to a matrix the size of c
    % minus c from the new point matrix
    % elementwise square the result
    % sum each square length along the second dimension
    % devide each length by its counterpart in the 2 * squared sigma matrix
    % get the exponent
    node = exp(-sum((repmat(ps(p,:), size(c,1), 1) - c).^2,2) ./ (2*s.^2));
    % add all of the results together and add the bias. (bias should
    % strictly be 1* b)
    n(p) = sum(node .* w) + b;
end
% ij reverses the y axis so the image is plotted in image space
% rather than standard x and y space
figure, hold on, axis ij square off, colormap jet
imagesc(reshape(n, N, N));
% figure
% colormap jet
% surf(reshape(n, N, N), 'EdgeColor','none','LineStyle','none','FaceLighting','phong')
% rotate3d
```

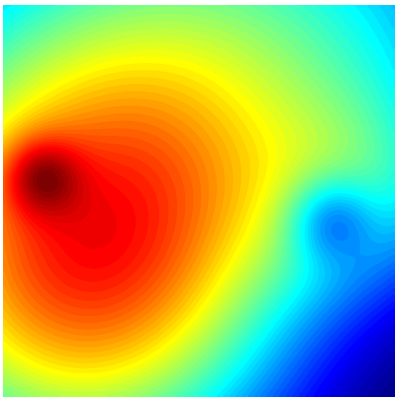

Figure 10: Visualised RBF Function

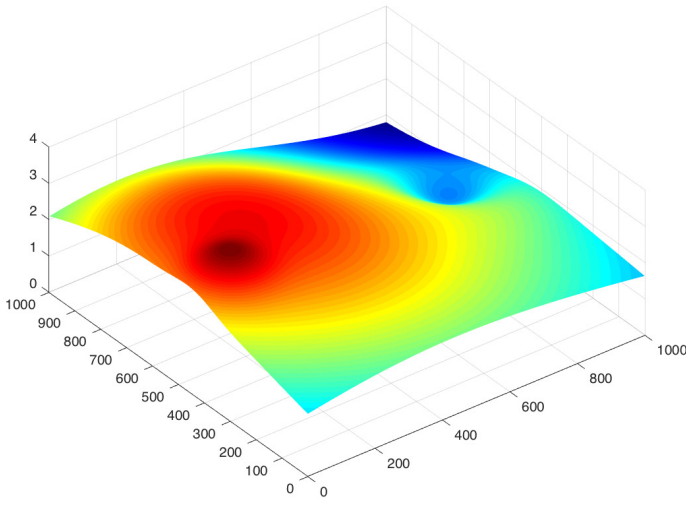

Figure 11: Visualised RBF Function Surface plot

Used a surface plot to further visualise the output of the funtion.