1 Task 1. Single Linkage Clustering

1.1 (a)

1.1.1 Solution

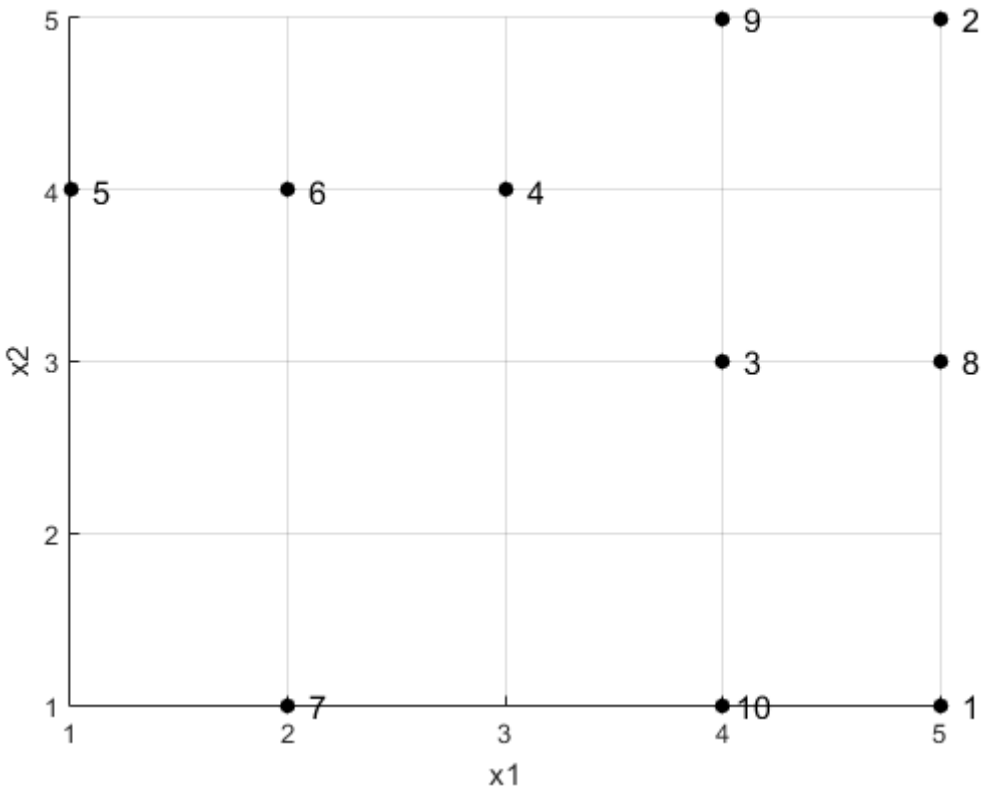

Figure 1: Solution for Task 1. (a)

1.1.2 Code

```
A = randi([1 5],10,2); % 10 objects with 2 features varying from 1 to 5

while size(unique(A,'rows'),1) ~= 10 % while there are not 10 different objects
    A = randi([1 5],10,2);
end

hold on;
grid on;
set(gca,'Xtick',1:5, 'Ytick',1:5);
xlabel('x1');
ylabel('x2');
plot (A(:,1),A(:,2), 'k.', 'MarkerSize', 20);
% converts the index of the point into a number and displays it
text(A(:,1)+0.06, A(:,2), cellstr(num2str((1:size(A,1))')),'FontSize',12);
```

1.2 (b)

1.2.1 Solution

Table 1: Solution for Task 1. (b)

0	5	5	2	1	4	2	10	8	10
5	0	18	9	2	1	5	5	9	17
5	18	0	9	8	13	5	17	9	5
2	9	9	0	5	10	8	20	18	20
1	2	8	5	0	1	1	5	5	9
4	1	13	10	1	0	2	2	4	10
2	5	5	8	1	2	0	4	2	4
10	5	17	20	5	2	4	0	2	4
8	9	9	18	5	4	2	2	0	2
10	17	5	20	9	10	4	8	2	0

1.2.2 Code

```
B = zeros(10,10);

for i = 1:size(A,1)
    for j = 1:size(A,1)
% for every element(i) check the distance against every other element(j)
        B(i,j) = sum((A(i,:) - A(j,:)).^2);
    end
end
```

1.3 (c)

1.3.1 Solution

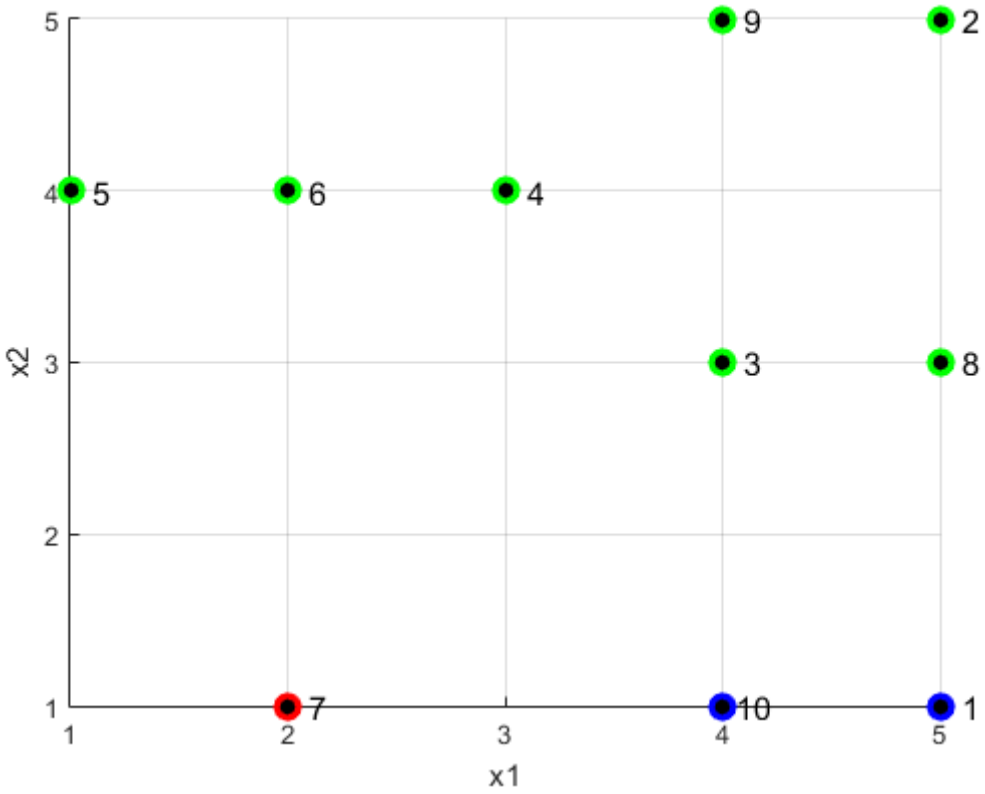

Figure 2: Solution for Task 1. (c)

1.3.2 Code

```
C = kruskals_mst_with_comments(A,3);

hold on;
grid on;
set(gca,'Xtick',1:5, 'Ytick',1:5);
xlabel('x1');
ylabel('x2');
plot(A(C==1,1), A(C==1,2), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'red');
plot(A(C==2,1), A(C==2,2), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'green');
plot(A(C==3,1), A(C==3,2), 'bo', 'MarkerSize', 10, 'MarkerFaceColor', 'blue');
% plots each point as the color of the label it represents
plot (A(:,1),A(:,2), 'k.', 'MarkerSize', 20);
text(A(:,1)+0.06, A(:,2), cellstr(num2str((1:size(A,1))')),'FontSize',12);
```

1.4 (d) & (e)

1.4.1 Solution

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

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

Figure 3: Solution for Task 1. (d) & Task 1. (e)

1.4.2 Code

```
function a = kruskals_mst_with_comments(b,c)
d = size(b,1); % number of points
e = nchoosek(1:d,2); % array with all edges (d*(d-1)/2 rows, 2 columns)
f = b(e(:,1),:) - b(e(:,2),:); % differences between point
% coordinates (these will be
% squared later to get the
% squared lengths)
[~,g] = sort(sum(f.*f,2)); % g is the index which will sort the
% edges from shortest to longest
e = e(g,:); % sorted edges according to length
a = 1:d; % forest with d trees (returned labels)
orig = 1:d; % save original data
step = 1; % step counter
fprintf('Step %d J = 0, #Cl %d: %s\n', step, numel(unique(a)), sprintf('( %d )',a))
% print the first step, with attributes and every cluster surrounded by '( )'
while numel(unique(a)) > c % check if the desired number of clusters
    % has been reached;
    % if not, go through the loop
    if a(e(1,1)) ~= a(e(1,2)) % if the vertices of the shortest
        % remaining edge are NOT in the same
        % cluster,
        a(a==a(e(1,1))) = a(e(1,2)); % relabel all vertices from
        % cluster A into cluster B
        % (This eliminates cluster A.)
        step = step + 1; % increment to the next step
        cDist(step) = sum(sqrt((e(1,:) - e(2,:)).^2));
        % get the criterion between the clusters to be joined
        uA = unique(a); % get the remaining amount of clusters,
        % to be used for indexing
        stepText(:,step) = sprintf('Step %d J = %d, #Cl %d: ', step, cDist(step), numel
        % stores then prints out the current step information
        fprintf(stepText(:,step));
        for i = 1:numel(unique(a)) % for the amount of elements in each cluster
            clusterData{step, i} = sprintf('( %s)', sprintf('%d ', orig(a == uA(i))));
            % stores the prints out each element, elements in each cluster are grouped
```

```
            % then surrounded by '( )' this is repeated until all clusters have been
            % processed
            fprintf(clusterData{step, i})
            % print out the current steps cluster information
        end
        fprintf('\n')
    end
    e(1,:) = []; % remove the shortest edge after use
end
mInd = max(find(cDist==max(cDist))) -1; % minus one to return step before highest
% finds the index with the maximum distance
% if there is a tie returns the one with the highest step index
a = cmunique(a) + 1;

fprintf('\nLargest jump == Step: %d >> return %s%s\n',mInd+1,...
    stepText(:,mInd), sprintf ('%s', clusterData{mInd,:}))
% returns information showing the largest step,
% and reprints the line of the step it should return

% as the labels for the c clusters may be
% any integers between 1 and d, "squeeze"
% them to be consecutive numbers 1, 2, 3,..., c
end
```

2 Task 2. k-means Clustering

2.1 (a)

2.1.1 Solution

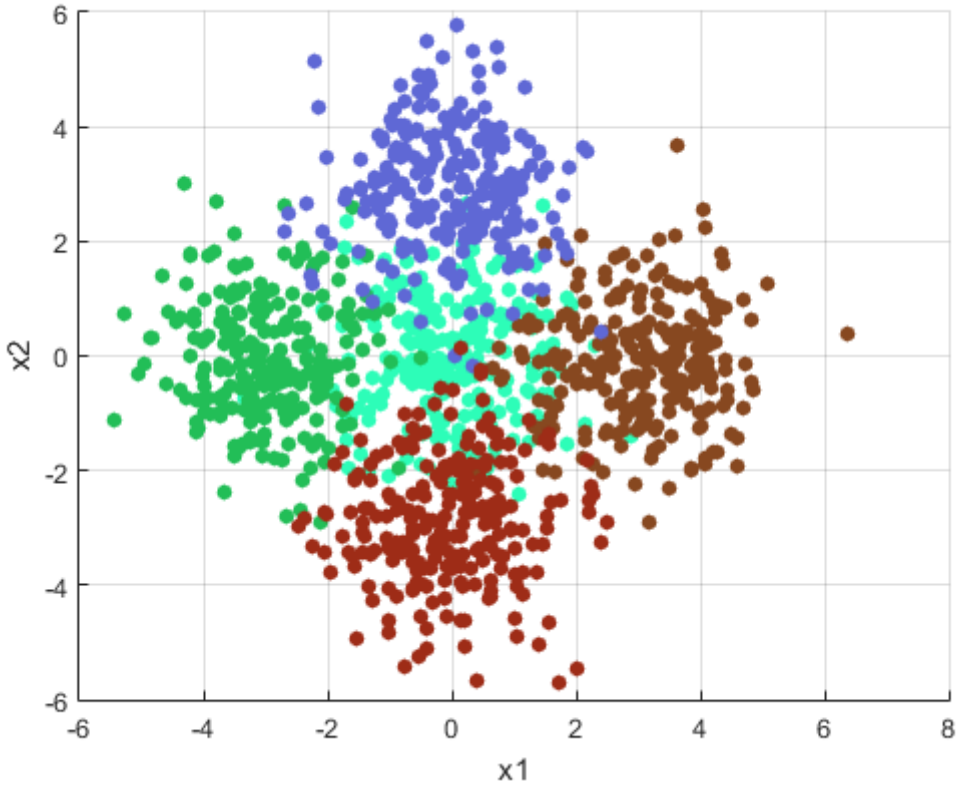

Figure 4: Solution for Task 2. (a)

2.1.2 Code

```
D = randn(1200,2);
labD = repmat((1:5)', numel(D(:,1))/5,1);
% repeats labels between 1 and 5, 240 times
D(labD==2,1) = D(labD==2,1) + 3;
D(labD==3,1) = D(labD==3,1) - 3;
D(labD==4,2) = D(labD==4,2) + 3;
D(labD==5,2) = D(labD==5,2) - 3;
% offsets the data depending on their label

hold on;
grid on;

for i = 1:numel(unique(labD)) % for the amount of labels
    plot(D(labD==i,1),D(labD==i,2),'.','color',rand(1,3),'MarkerSize',20)
    % plot the current data with that label using a random colour
end
```

2.2 (b)

2.2.1 Solution

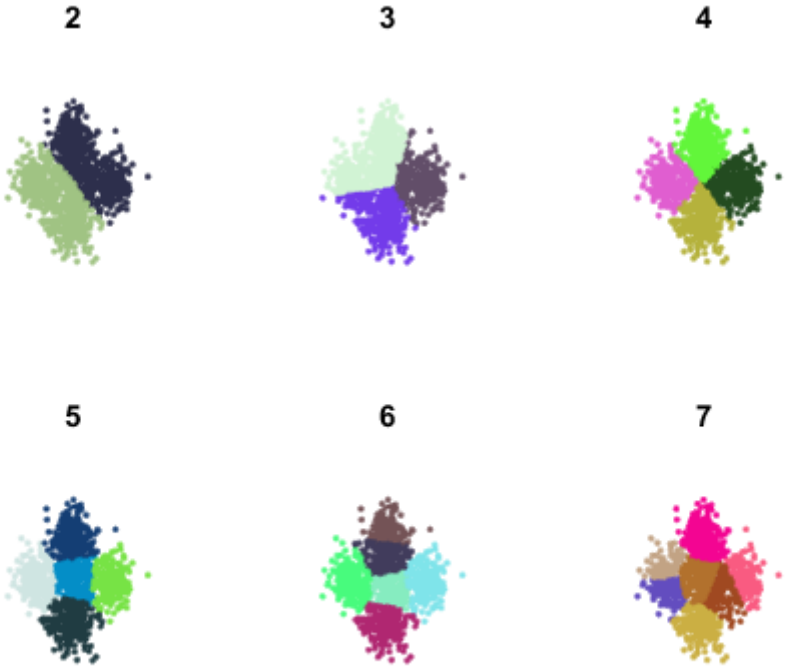

Figure 5: Solution for Task 2. (b)

2.2.2 Code

```
for i = 2:7 % amount of different cluster inputs to plot
    subplot(2, 3, i-1);
    title(i);
    axis off;
    hold on;
    E = kmeans_cg(D,i);
    % enters function with i clusters as target
    for j = 1:i % amount of current clusters
        plot(D(E==j,1),D(E==j,2),'.','color',rand(1,3))
        % plot that cluster with a random color
    end
end
```

3 Task 3. Feature Selection and Extraction

3.1 (a)

3.1.1 Code

```
load 'Example_MNIST_digits.mat'

D38 = b(labb==4 | labb==9,:);
% only digits 3 and 8
D38lab = labb(labb==4 | labb==9);
% corresponding labels
```

3.2 (b)

3.2.1 Solution

```
                                        Top 10 Features

Index ::          488        489        462        461        516        515        490        463        517        487

Criterion Value ::  149.102804  148.479389  141.630909  139.663063  138.486085  130.648703  122.048336  119.217545  117.486386  117.240697
```

Figure 6: Solution for Task 3. (b)

3.2.2 Code

```
F = sqrt((mean(D38(D38lab == 4,:),1) - mean(D38(D38lab == 9,:),1)).^2);
% criterion value by the euclidean distance
% between the means of the features for both classes
[G, indG] = sort(F,'descend');
% sort the values from highest value to lowest
% and get the index
D38 = D38(:,indG);
% sort the features of the dataset from highest to lowest
% using the index of the criterion sort

fprintf('\t\t\t\t\t\t\t\t\t\t\t\tTop 10 Features\n\nIndex ::');
fprintf('\t\t\t%d', indG(1:10));
fprintf('\n\nCriterion Value ::');
fprintf('\t%f', G(1:10));
% displays the index of that feature and the criterion value
```

3.3 (c)

3.3.1 Solution

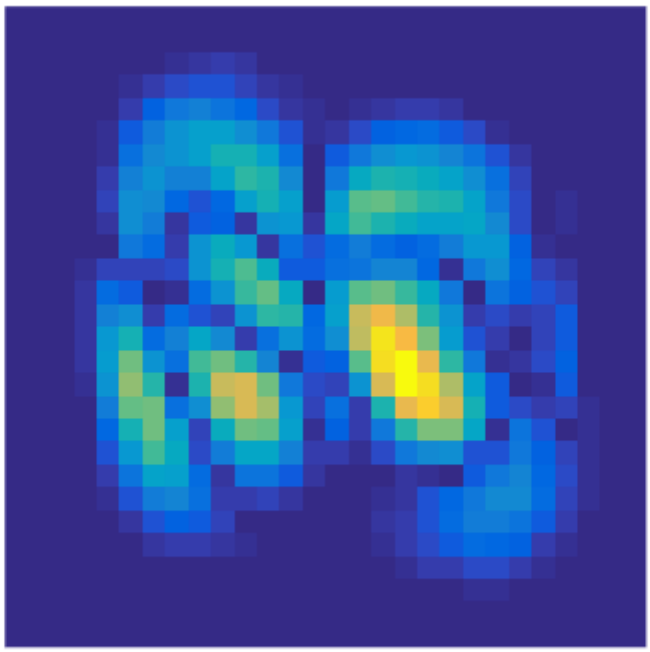

Figure 7: Solution for Task 3. (b)

3.3.2 Code

```
hold on;
axis square;
axis off;
imagesc(reshape(F,28,28))
% reshapes the criterion values to have 28 rows and 28 cols
```

3.4 (d)

3.4.1 Solution

```
Number of Irrelevant Values ::  235
```

Figure 8: Solution for Task 3. (d) - Number of Irrelevant Values

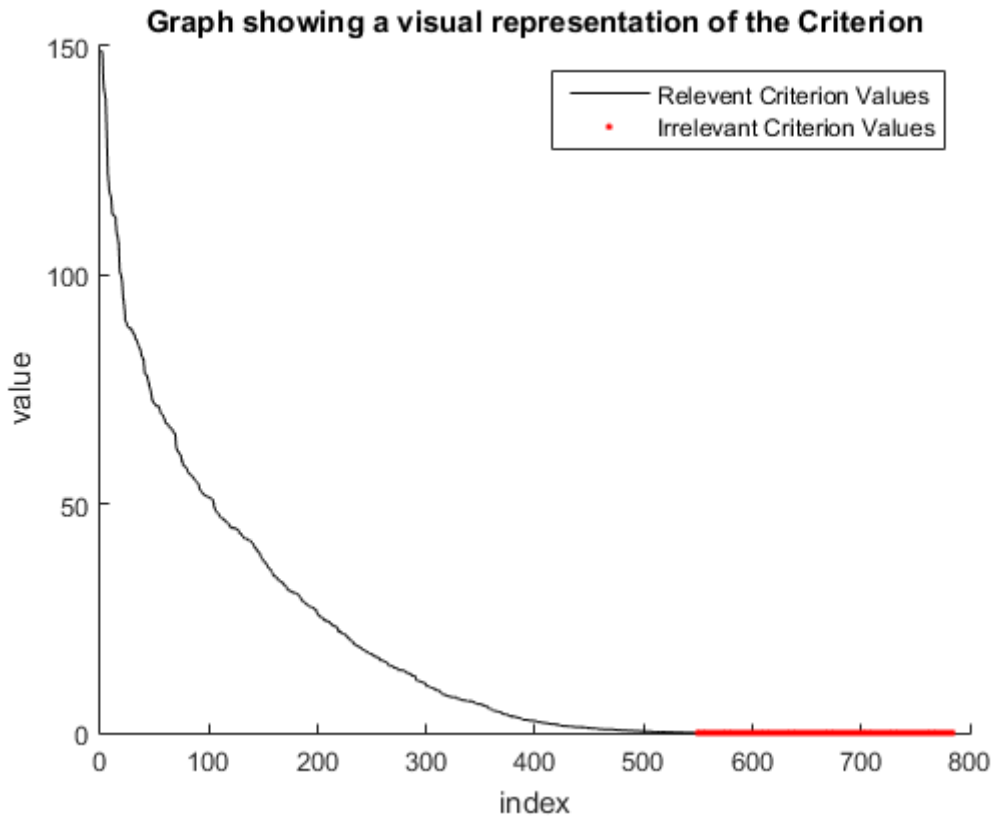

Figure 9: Solution for Task 3. (d) - Representation of the Criterion

3.4.2 Code

```
fprintf('Number of Irrelevant Values ::\t%d\n', numel(G(G==0)));
% number of values which are equal to 0

hold on;
xlabel('index');
ylabel('value');
title('Graph showing a visual representation of the Criterion');

plot(G, 'k');
plot(find(G==0),0,'r.');
% where the value is equal 0 plot a red dot

legend('Relevent Criterion Values','Irrelevant Criterion Values');

D38_clean = D38(:,G~=0);
% containing values which are not equal to 0
```

3.5 (e)

3.5.1 Solution

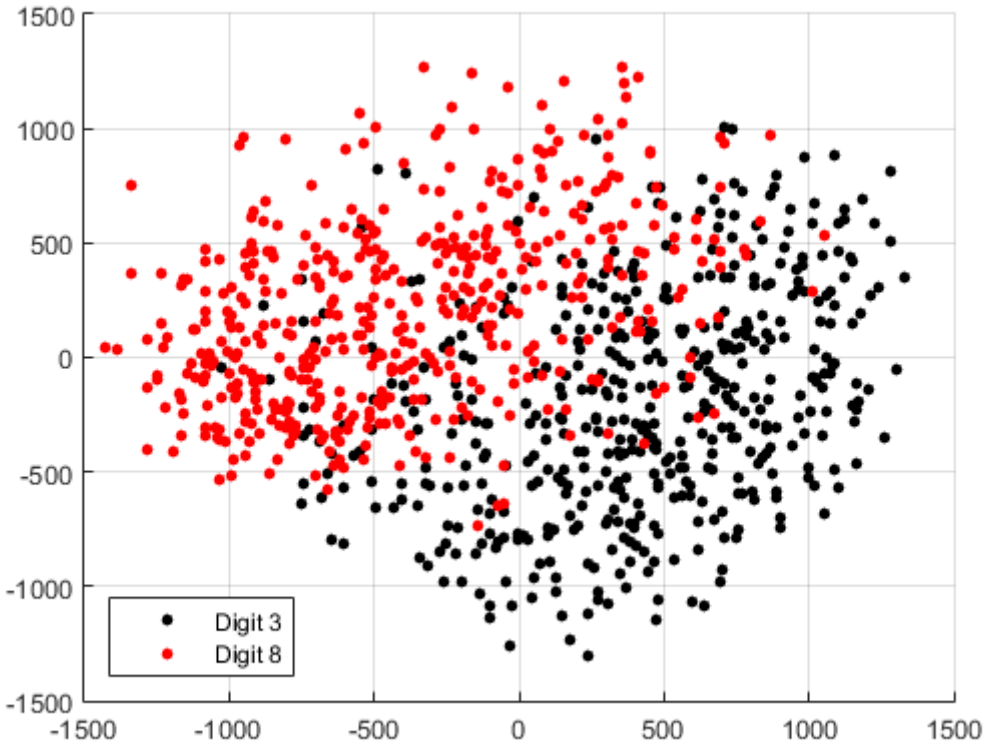

Figure 10: Solution for Task 3. (e)

3.5.2 Code

```
[~, pc] = pca(D38_clean);

hold on;
grid on;

plot(pc(D38lab == 4,1), pc(D38lab == 4,2), '.k', 'MarkerSize', 15);
plot(pc(D38lab == 9,1), pc(D38lab == 9,2), '.r', 'MarkerSize', 15);
% the first pca of each digit plotted against the second pca of each digit

legend('Digit 3', 'Digit 8', 'Location', 'southwest');
```

3.6 (f)

3.6.1 Solution

```
D38 NMC Error = 0.144351
PC NMC Error = 0.135983
Difference = 0.008368
```

Figure 11: Solution for Task 3. (f)

Overall the error rates of both datasets are very low. PC has a slightly lower error rate, this is because the PCA manages to find more characteristics about the feature values. Meaning there are more unique values in the data-set to differentiate between. Instead of having duplicate or similar values which make the means of the classes closer to each other, making it harder to classify correctly.

3.6.2 Code

```
J = my_nmc(D38(1:round(size(D38,1)/2),1:8),...
    D38lab(1:round(size(D38lab)/2)), D38(round(size(D38,1)/2)+1:end,1:8));
K = sum(D38lab(round(size(D38lab)/2)+1:end) ~= J)/numel(J);

L = my_nmc(pc(1:round(size(D38,1)/2),1:8), ...
    D38lab(1:round(size(D38lab)/2)), pc(round(size(D38,1)/2)+1:end,1:8));
M = sum(D38lab(round(size(D38lab)/2)+1:end) ~= L)/numel(L);
% selects the first 8 features, splits the dataset and labels in half
% first half for training second half for testing
fprintf('D38 NMC Error = %f\\n', K);
fprintf('PC NMC Error = %f\n', M);
fprintf('Difference = %f\n', K - M);
```

4 Task 4. Perceptron

4.1 Solution

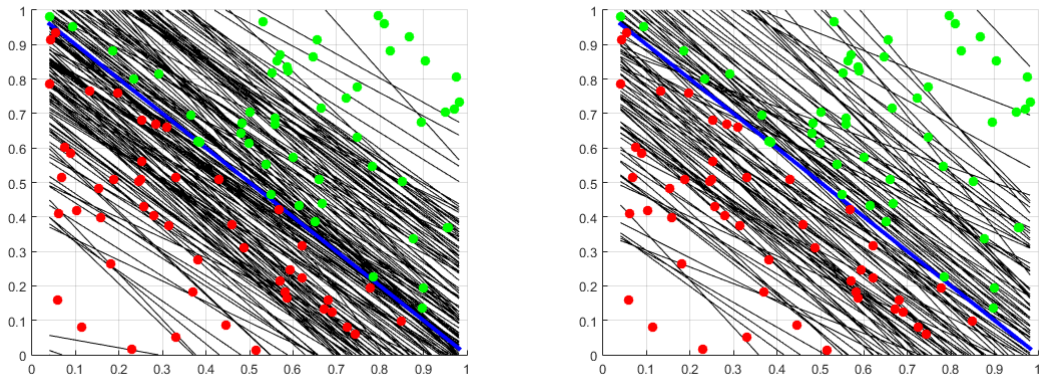

Figure 12: Solution for Task. 4 - (Left)$\eta = 0.2, (Right)\eta = 1$

The outputs show that in general the higher the learning rate the less epochs the perceptron needs to get an error free function. However the difference between the number of epochs required can vary depending on how lucky you might get with the initial randomization of the weights.

4.2 Code

4.2.1 Main

```
N = 1*rand(100,2);
labN =  0.4 * N(:,1) + 0.4 * N(:,2) - 0.4 > 0;
% creates two classes if result of point is greater than function
% then class = 1, if not class = 0
O = perceptron(N, labN, 0.2);
O = perceptron(N, labN, 1);
% runs the script twice for different values of eta
```

4.2.2 perceptron.m

```
function w = perceptron(d,l,eta)
[p q] = size(d);
w = rand(q+1,1);
% assigns 3 random weights bias + x & y
X = [min(d(:,1)), max(d(:,1))];
% vector containing the min
% and max x values
figure;
hold on;
grid on;
axis([0 1 0 1]);
Y = ((w(2) * -1) * X + (w(1) * -1))/w(3);
% the random weights of this epoch plugged into function
% to create y vector
plot(X, Y, 'k'); % draw epoch function
while q*q' % feature * feature
    for i = 1:p % for the amount of objects
        z = [1 d(i,:)]; % bias + x & y to plug against weights
        v = z * w < 0;
        q(i) = v==l(i);
        w = w + q(i)*(2*v-1)*eta*z';
        Y = ((w(2) * -1) * X + (w(1) * -1))/w(3);
        % the calculated weights of this epoch plugged into function
        labN =  w(2) * d(:,1) + w(3) * d(:,2) + w(1) > 0;
        plot(X, Y, 'k');
        % draw epoch function
    end
end
    Y = (-0.4 * X + 0.4)/0.4;
    % y vector for the function that
    % split the classes
    plot(X, Y, 'b','Linewidth',3);
    % plot the function
    % this is the function the perceptron
    % is aiming for
    plot(d(labN == 0,1), d(labN == 0,2), 'r.', 'MarkerSize', 25);
    plot(d(labN == 1,1), d(labN == 1,2), 'g.', 'MarkerSize', 25);
    % plot classes based on which side of the line they are on
    % could move the above into the loop with a pause to update classes
    % every epoch outside works faster for end result
end
```

5 Task 5. RBF

5.1 Solution

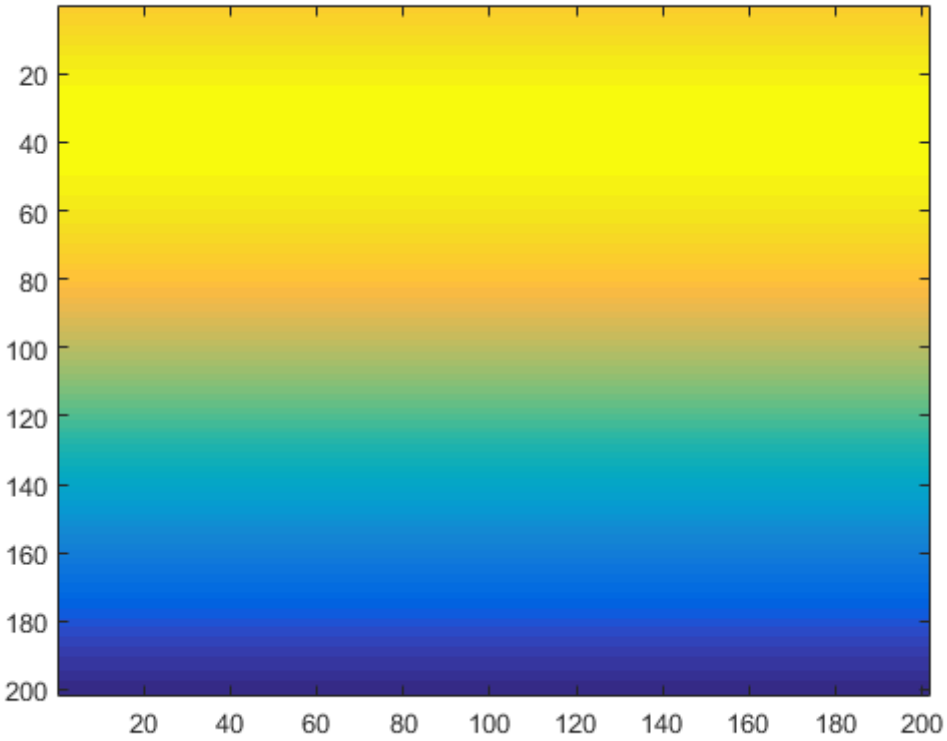

Figure 13: Solution for Task 5.

5.2 Code

```
[xx, yy] = meshgrid(0:0.005:1, 0:0.005:1);
% creating pairs of points between 0 and 1

cx = repmat([0.84, 0.49, 0.67, 0.92, 0.94, 0.22, 0.21, 0.81,...
    0.10, 0.21]', 1, 201);
cy = repmat([0.55, 0.61, 0.25, 0.59, 0.18, 0.30, 0.70, 0.84,...
    0.44 0.07]', 1, 201);
S = repmat([0.08, 0.70, 0.59, 0.18, 0.30, 0.70, 0.84, 0.44,...
    0.07, 0.41]', 1, 201);
w = repmat([-0.64, 0.51, 1.06, 0.51, -0.35, -1.49, 1.53, -0.50,...
    0.58, 1.89]', 1, 201);
% creating centres sigmas and weights for every data point in the grid

P = zeros(201, 201);

for i = 1:201
        x = repmat(xx(i,:), 10, 1);
        y = repmat(yy(i,:), 10, 1);
        % ith row of xx and yy on the grid
    for j = 1:10
        P(i, :) = P(i, :) + (exp(-(((x(j,:) - cx(j,:)).^2 ...
            + (y(j,:) - cy(j,:)).^2) / (2 * S(j,:) .^2)))) * w(j,:);
        % sum of the activation function for each class separately
        % (I think this is where I went wrong)
    end
end

imagesc(P);
```