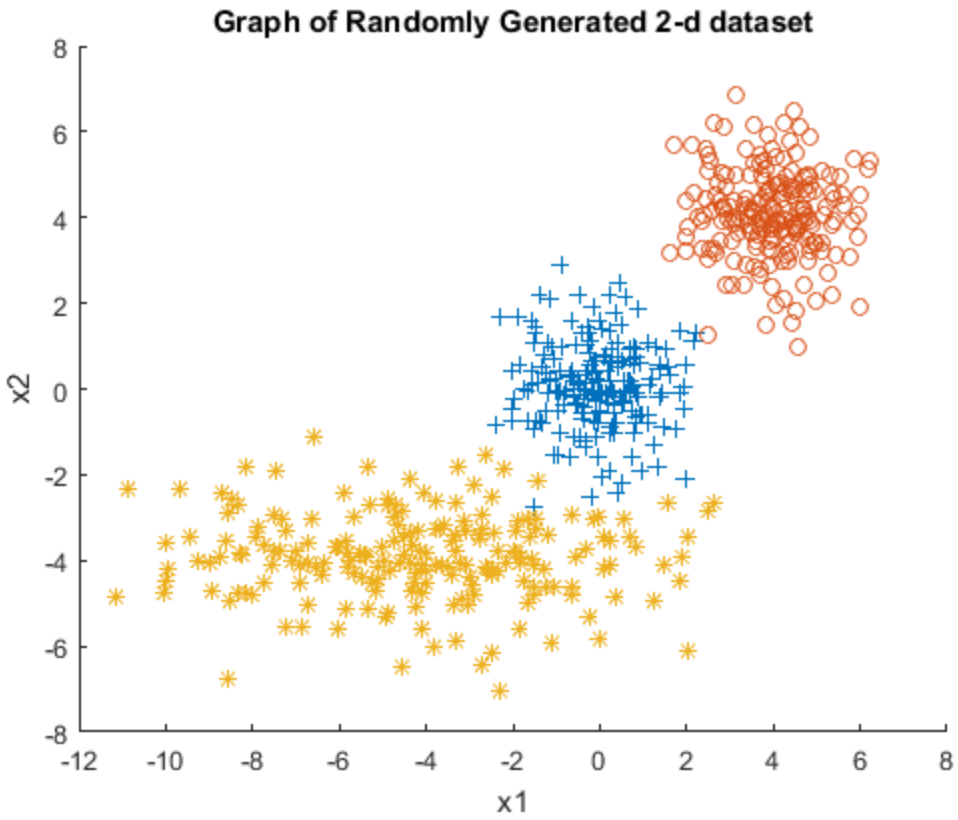
Contents


```
% Initialization
clear all;
close all;
clc;
```

Create Data as instructed in the assignment

2-D datasets using matlab's Gaussian random number generator randn

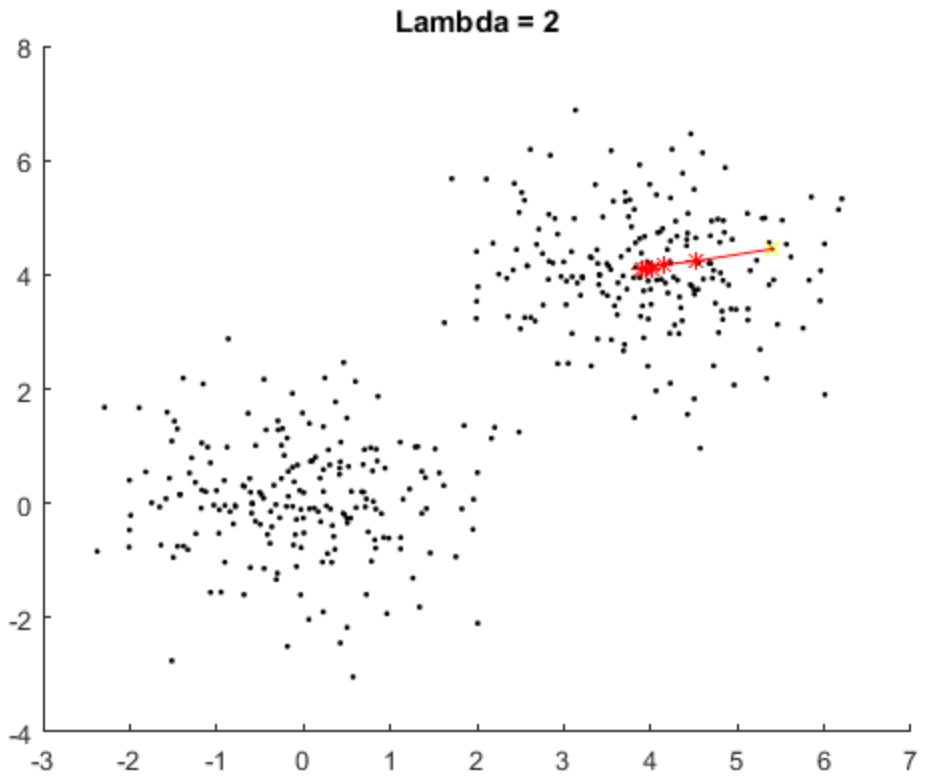
```
a = randn(200, 2);
b = a + 4;
c = a;
c(:, 1) = 3 * c(:, 1);
c = c - 4;
d = [a; b];
e = [a; b; c];
figure; hold on;
title('Graph of Randomly Generated 2-d dataset')
xlabel('x1')
ylabel('x2')
plot(a(:, 1), a(:, 2), '+');
plot(b(:, 1), b(:, 2), 'o');
plot(c(:, 1), c(:, 2), '*');
hold off;
```



P4Q1: Mean Shift Clustering Algorithm Implementation

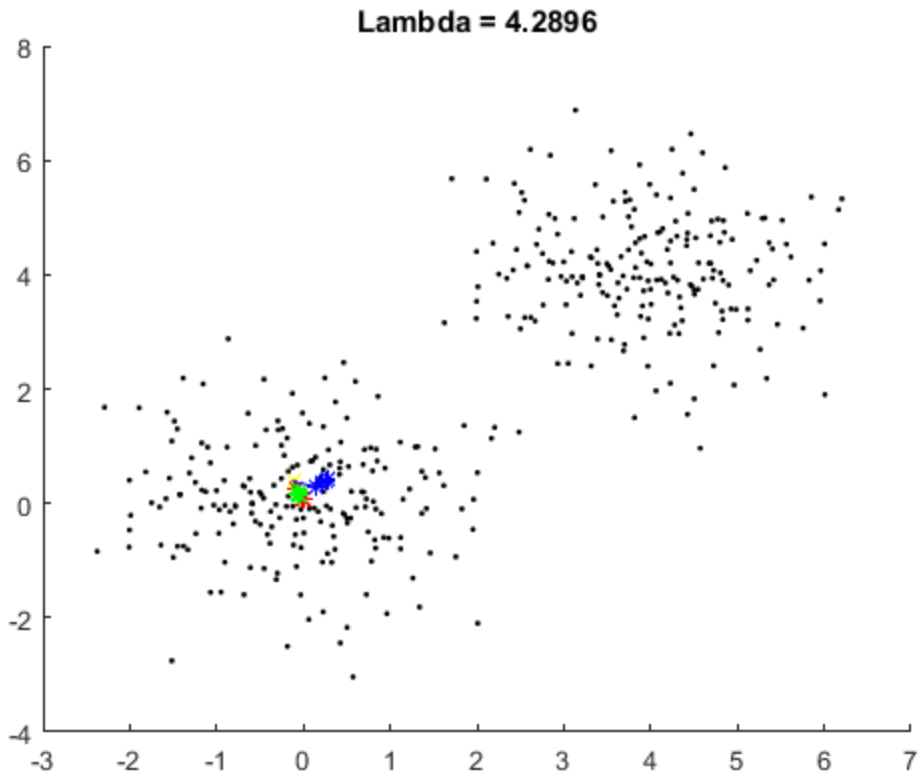
See Mean Shift Clustering Function and Flat Kernel Function for more details

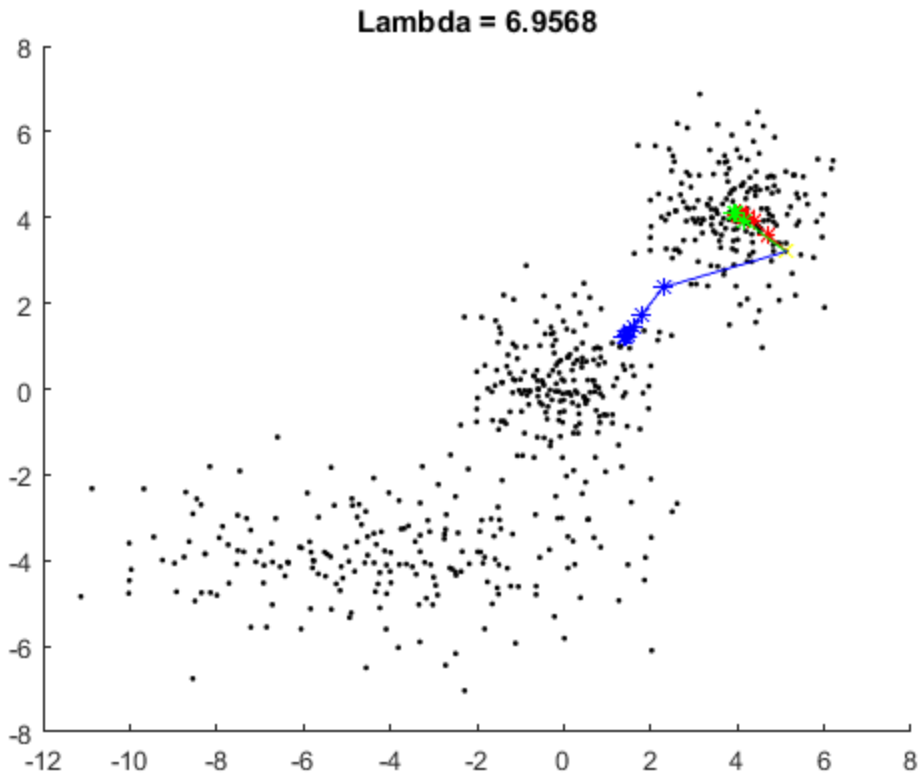
```
lambda = 2;
mx = meanShift(d, lambda);
```

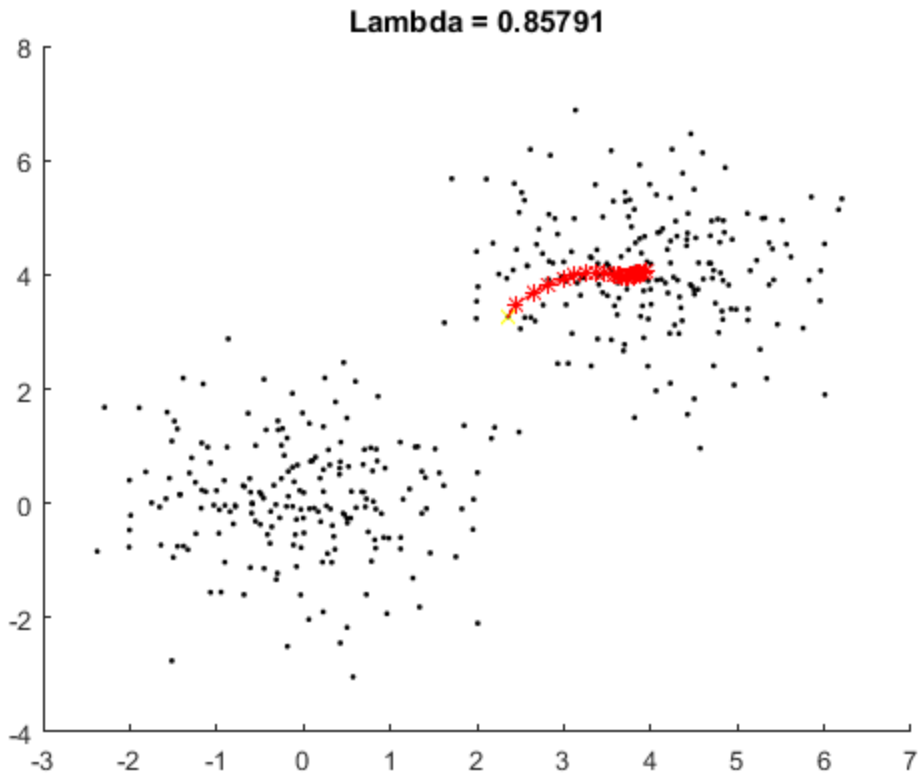


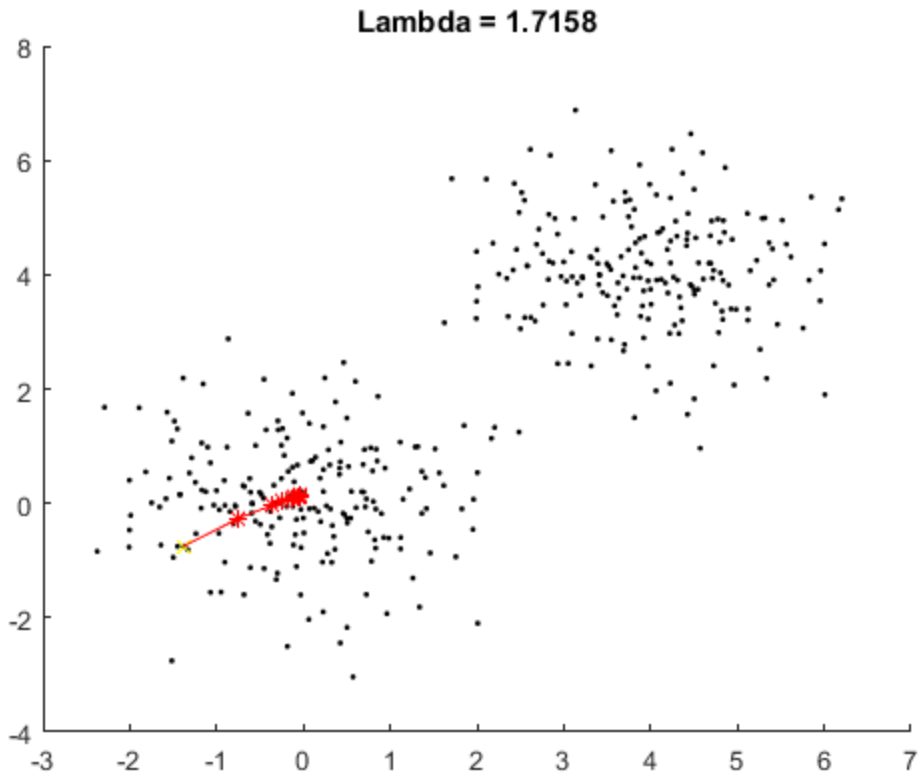
P4Q2: Cluster Centres Plot

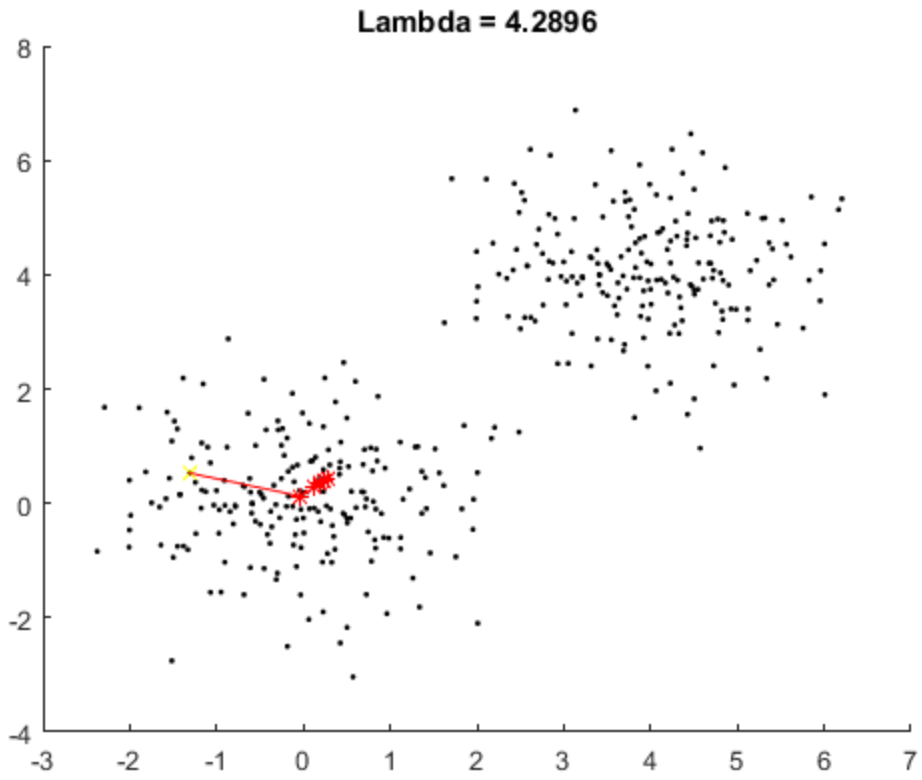
```
L = 3;
Rd = min(range(d));
lambdad = [0.1*Rd 0.2*Rd 0.5*Rd];
Re = min(range(e));
lambdae = [0.1*Re 0.2*Re 0.5*Re];
Mxd = meanShift(d, lambdad);
Mxe = meanShift(e, lambdae);
for l = 1:L;
    meanShift(d, lambdad(l));
end
for l = 1:L;
    meanShift(e, lambdae(l));
end

% Q2 Comment
% The higher the lambda value, the better the result is. Because the step size gets
% large. Besides, we reach a place with higher density thus achieving a
% better maxima. However very large value of lambda also can make the
% result differ completely.
```

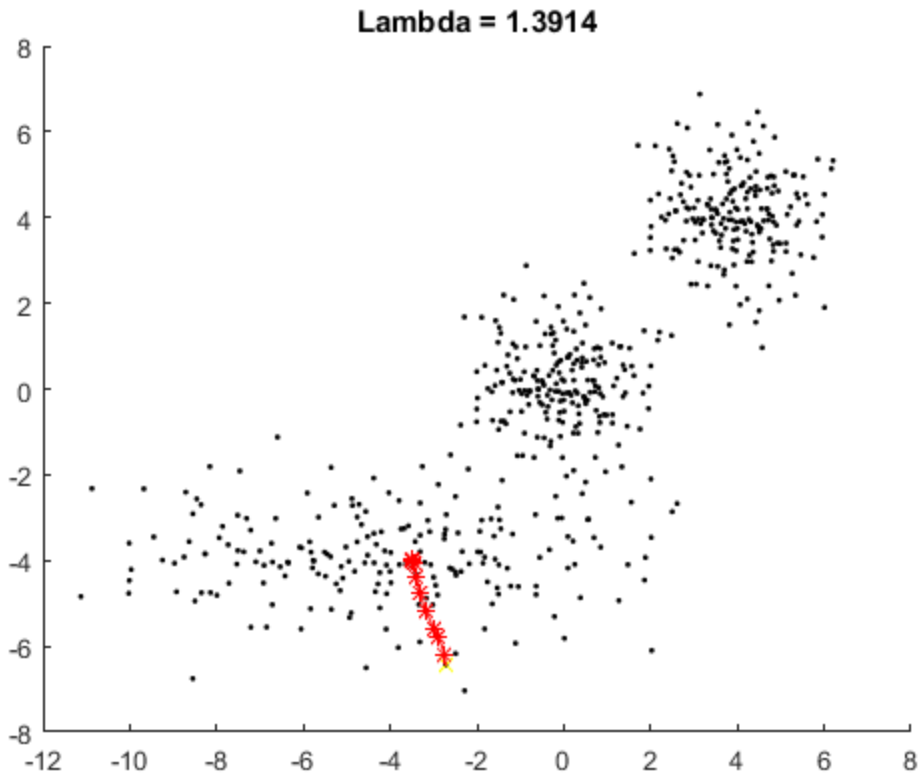

Lambda = 4.2896
8
6
4
2
0
-2
-4
-3
-2
-1
0
1
2
3
4
5
6
7

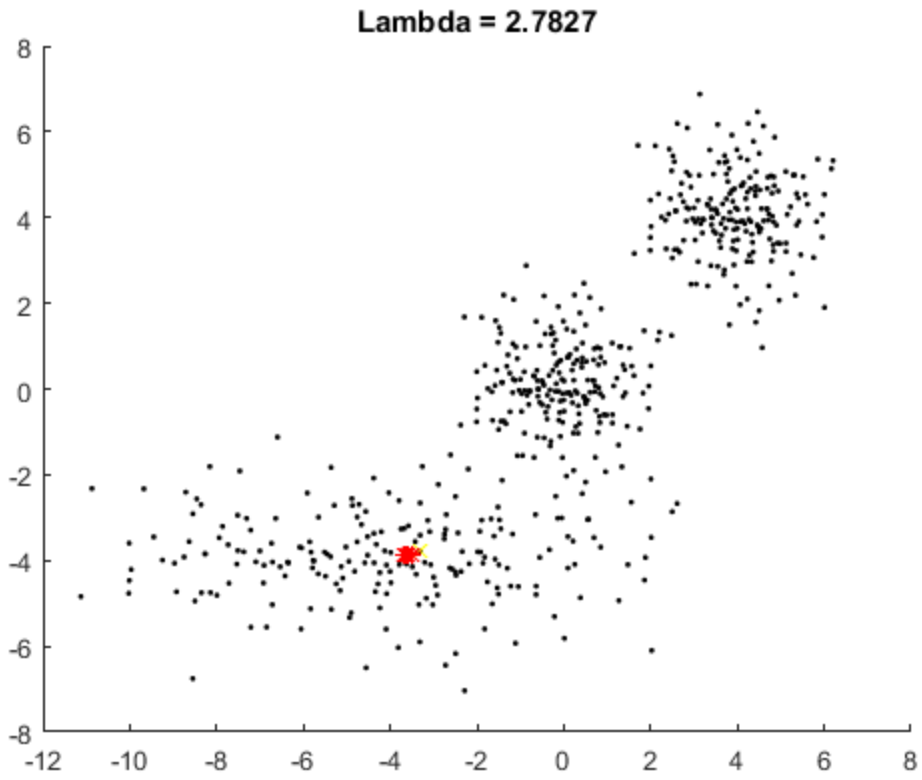

Lambda = 6.9568
8
6
4
2
0
-2
-4
-6
-8
-12
-10
-8
-6
-4
-2
0
2
4
6
8

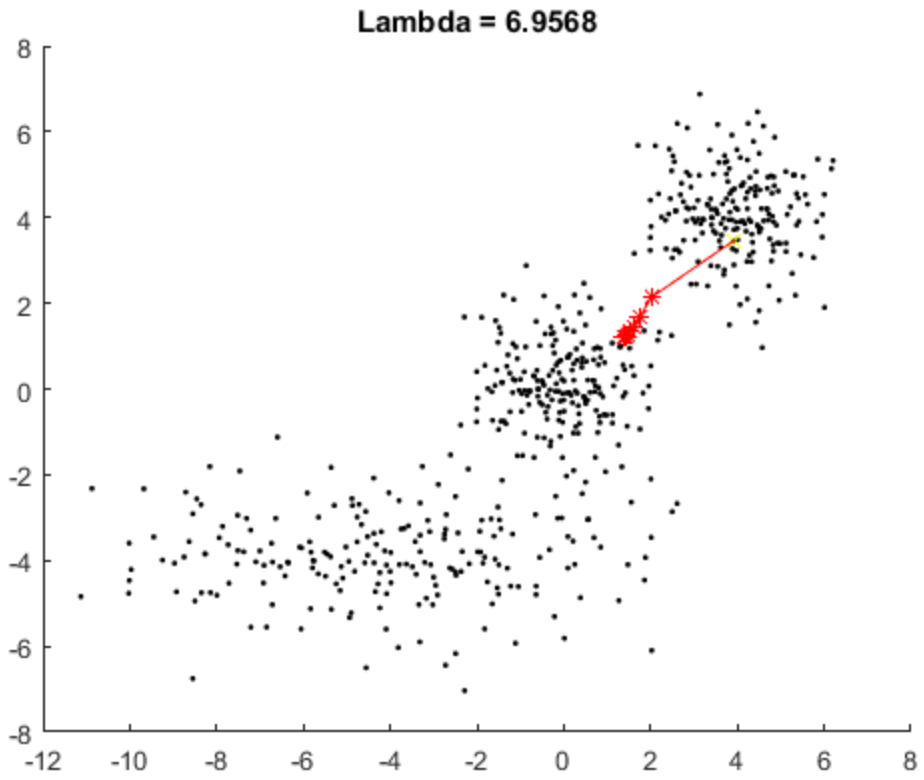

Lambda = 0.85791
8
6
4
2
0
-2
-4
-3
-2
-1
0
1
2
3
4
5
6
7


Lambda = 1.7158
8
6
4
2
0
-2
-4
-3
-2
-1
0
1
2
3
4
5
6
7


Lambda = 4.2896
8
6
4
2
0
-2
-4
-3
-2
-1
0
1
2
3
4
5
6
7


Lambda = 1.3914
8
6
4
2
0
-2
-4
-6
-8
-12
-10
-8
-6
-4
-2
0
2
4
6
8


Lambda = 2.7827
8
6
4
2
0
-2
-4
-6
-8
-12
-10
-8
-6
-4
-2
0
2
4
6
8



Published with MATLAB® R2015b

Contents

Initialization

```
clear all;
close all;
clc;
```

Data Preperation

Loading Pima Diabetes dataset for Q2

```
fid = fopen('pima_indians_diabetes.csv', 'r');
textscan(fid, '%f,%f,%f,%f,%f,%f,%f,%f,%s', 1);
raw = textscan(fid, '%f,%f,%f,%f,%f,%f,%f,%f,%s');
X = cell2mat(raw(1:end-1));
y = cell2mat(raw{end})==repmat('pos', size(X, 1), 1);
y = double(y(:, 1));
fclose(fid);
```

P5Q1: Eigen Function Call Implementing Principal Component Analysis

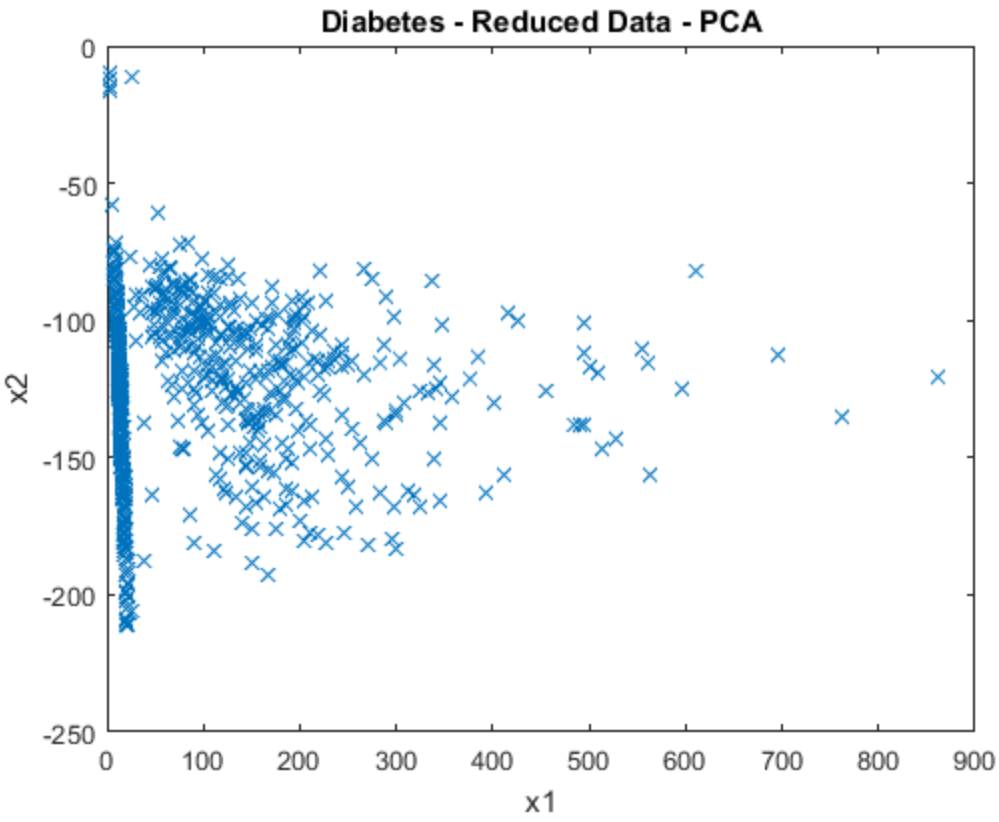
```
[U, s] = eigen(X);
```

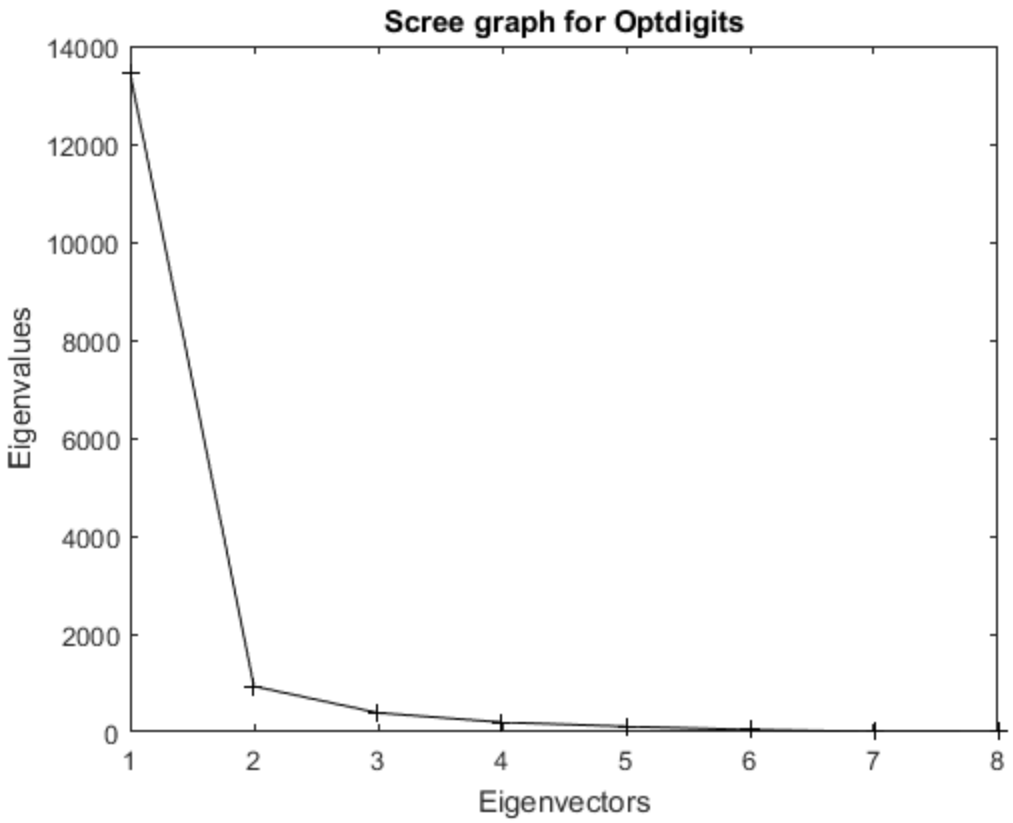
P5Q2: Principal Component Analysis

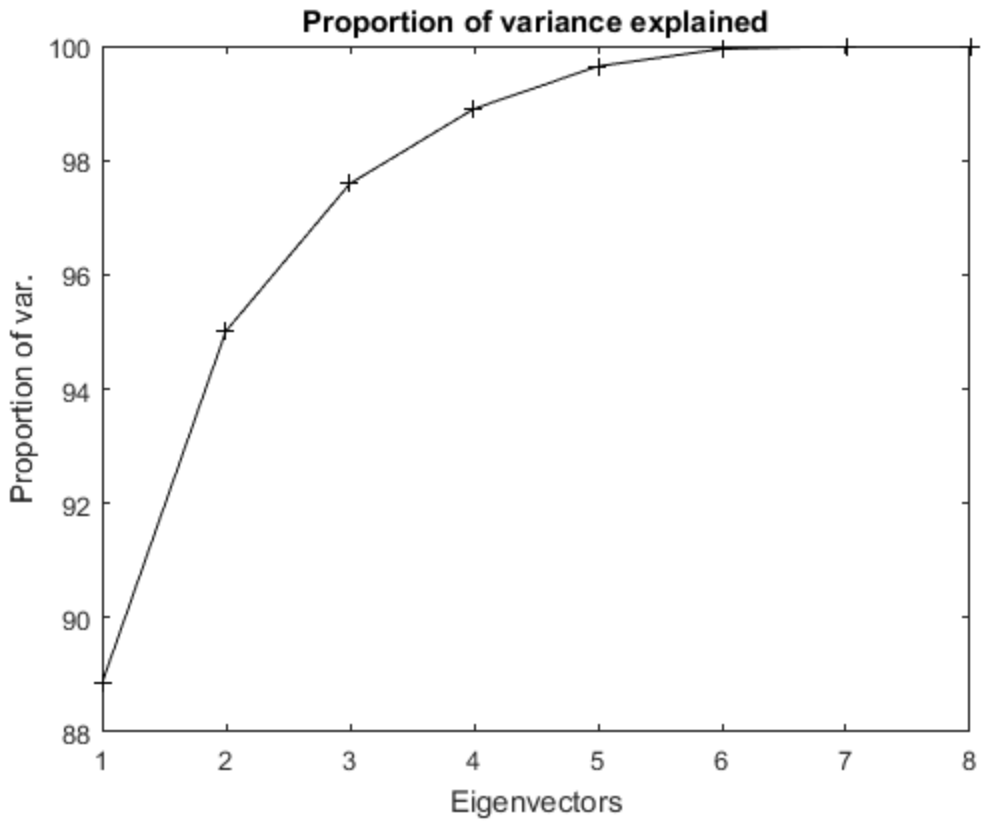
```
k = 2;
Ur = U(:, 1:k);
sr = s(1:k);
Xr = X*Ur;
% P5Q2a) plot of the data in the space spanned by the first two principal
% components
plot(Xr(:, 1), Xr(:, 2), 'x');
xlabel('x1');
ylabel('x2');
```

```
title('Diabetes - Reduced Data - PCA');
% P5Q2b) percentage of the data variance
ret = sum(sr)/sum(s)*100;
fprintf('percentage of the data variance = %f\n', ret);
% P5Q2c) Scree graph
figure; plot(s, 'k+-');
xlabel('Eigenvectors');
ylabel('Eigenvalues');
title('Scree graph for Optdigits');
figure; plot(cumsum(s)/sum(s)*100, 'k+-');
xlabel('Eigenvectors');
ylabel('Proportion of var.');
title('Proportion of variance explained');
```

```
percentage of the data variance = 95.013742
```

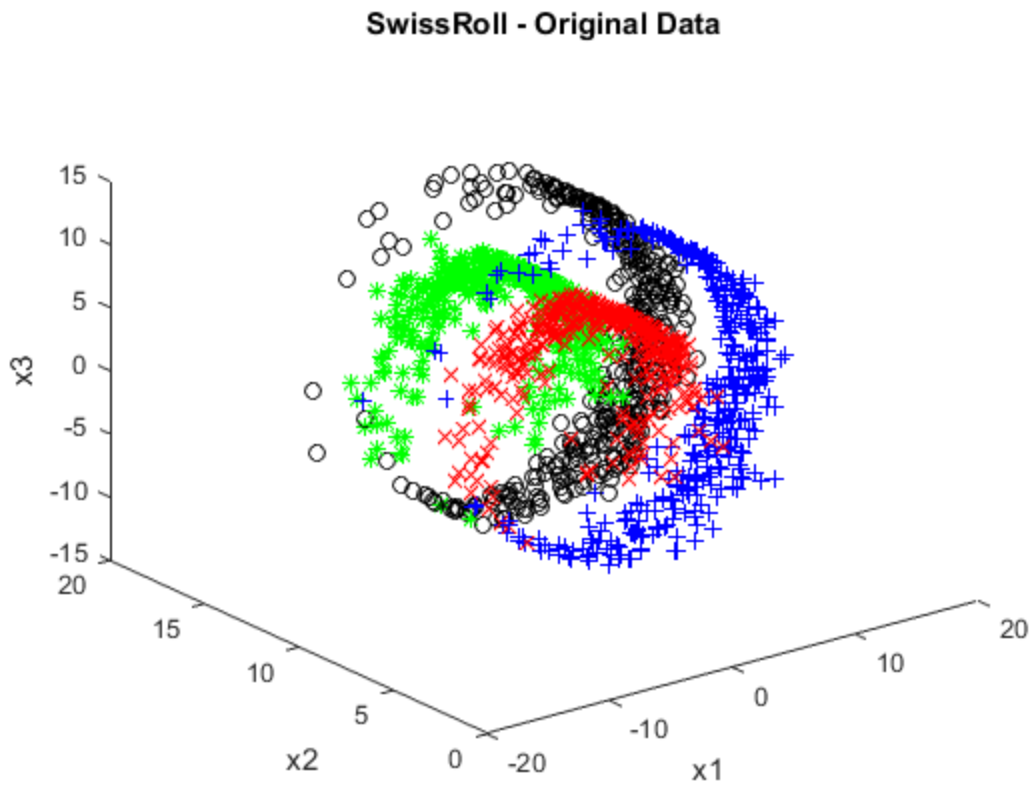

Diabetes - Reduced Data - PCA
0
-50
-100
-150
-200
-250
x2
0
100
200
300
400
500
600
700
800
900
x1


Scree graph for Optdigits
14000
12000
10000
8000
6000
4000
2000
0
Eigenvalues
1
2
3
4
5
6
7
8
Eigenvectors



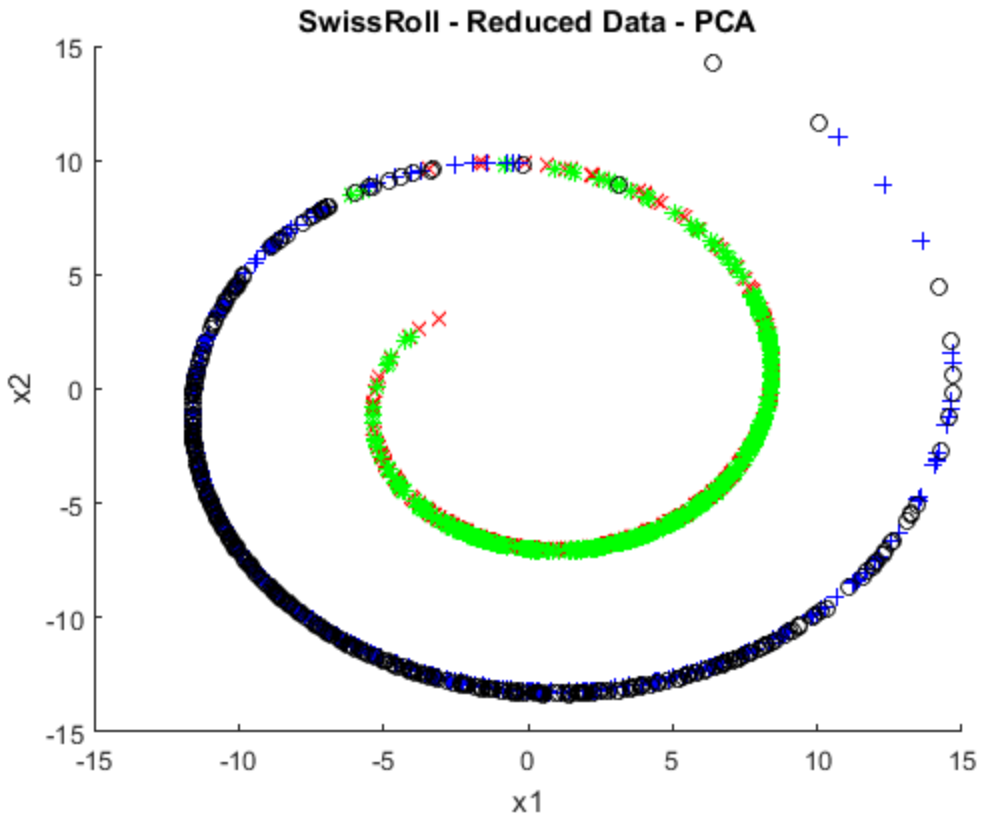
P5Q3: plot of the swissroll dataset in 3D

```
X = load('swissroll.dat');
figure; hold on;
plot3(X(1:400, 1), X(1:400, 2), X(1:400, 3), 'rx');
plot3(X(401:800, 1), X(401:800, 2), X(401:800, 3), 'g*');
plot3(X(801:1200, 1), X(801:1200, 2), X(801:1200, 3), 'b+');
plot3(X(1201:1600, 1), X(1201:1600, 2), X(1201:1600, 3), 'ko');
xlabel('x1');
ylabel('x2');
zlabel('x3');
title('SwissRoll - Original Data');
hold off;
view (3);
```



P5Q4: PCA function on the Swiss Roll

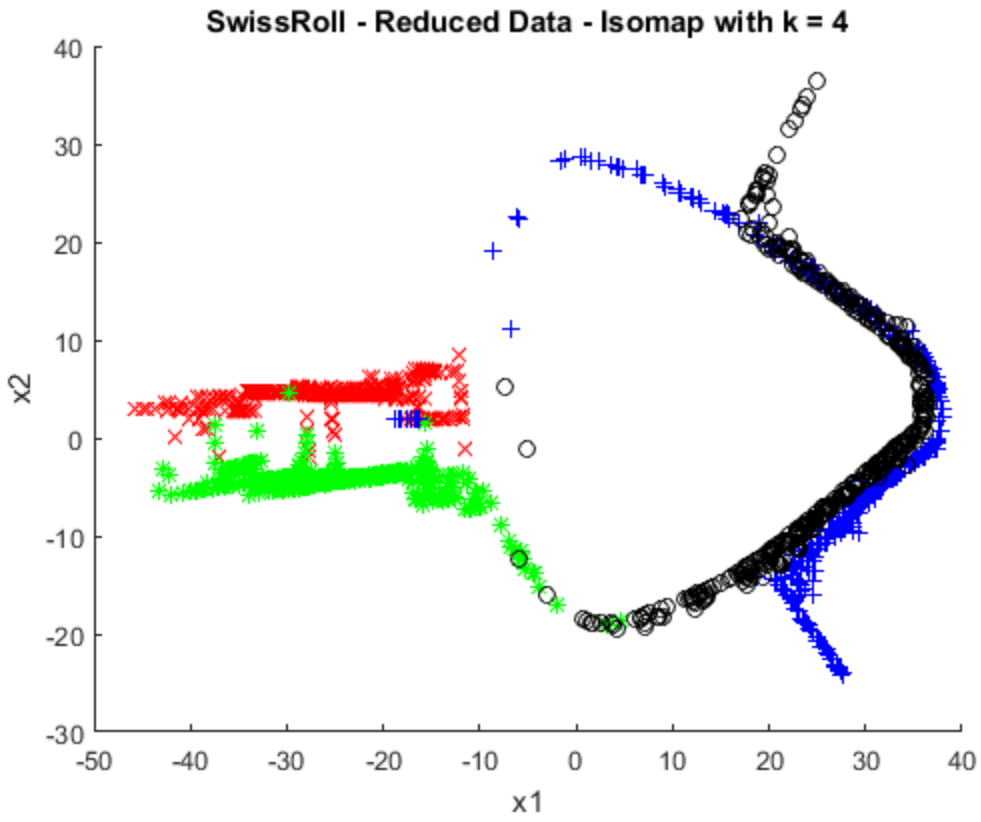
```
[U, s] = eigen(X);
k = 2;
Ur = U(:, 1:k);
Xr = X*Ur;
% P5Q4a) plot of the data in the space spanned by the first two principal
% components
figure; hold on;
plot(Xr(1:400, 1), Xr(1:400, 2), 'rx');
plot(Xr(401:800, 1), Xr(401:800, 2), 'g*');
plot(Xr(801:1200, 1), Xr(801:1200, 2), 'b+');
plot(Xr(1201:1600, 1), Xr(1201:1600, 2), 'ko');
xlabel('x1');
ylabel('x2');
title('SwissRoll - Reduced Data - PCA');
hold off;
% P5Q4b) Comment on PCA's ability to separate the clusters
% In this case, PCA wasn't able to seperate those clusters which had a
% different third dimension (the one we dropped).
```

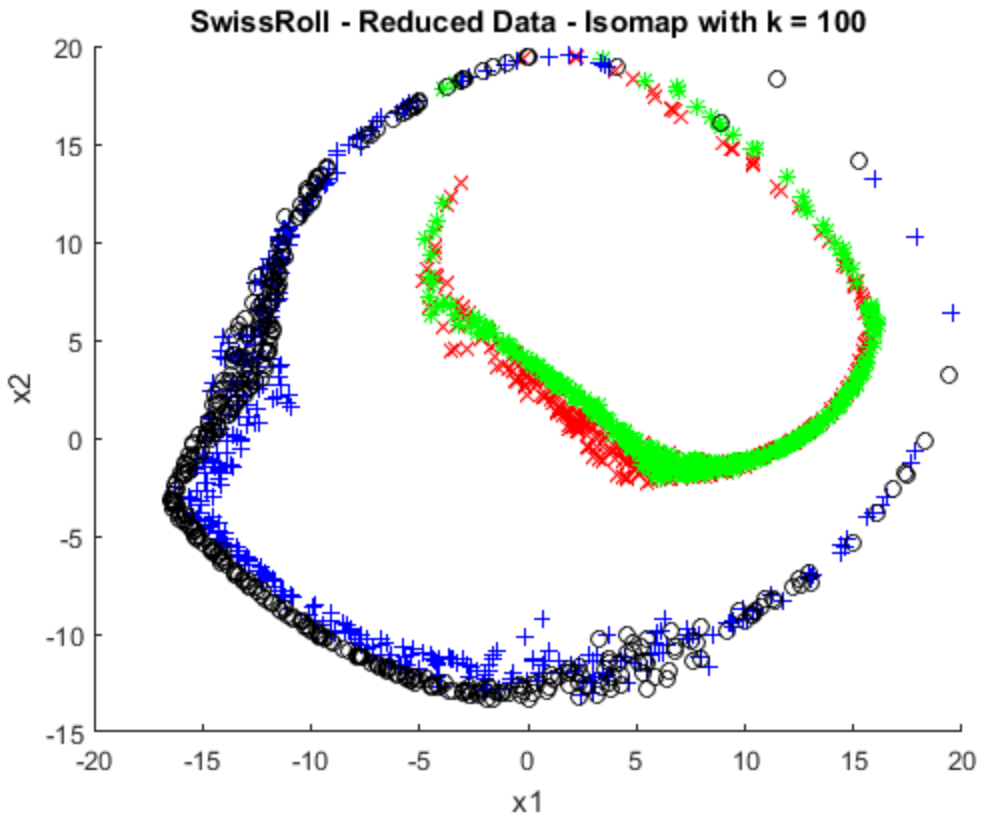


P5Q5: Run Isomap to reduce the data to 2D.

```
m = size(X, 1);
D = zeros(m, m);
d = pdist(X, 'euclidean')';
b = ~triu(ones(m, m));
D(b) = d;
clear b d;
D = D + D';
k = 4;
dim = 2;
Xr = isomap(D, dim, k)';
% P5Q5a)
figure; hold on;
plot(Xr(1:400, 1), Xr(1:400, 2), 'rx');
plot(Xr(401:800, 1), Xr(401:800, 2), 'g*');
plot(Xr(801:1200, 1), Xr(801:1200, 2), 'b+');
plot(Xr(1201:1600, 1), Xr(1201:1600, 2), 'ko');
xlabel('x1');
ylabel('x2');
title('SwissRoll - Reduced Data - Isomap with k = 4');
hold off;
```

```
% P5Q5b) Isomap vs PCA
% Clusters (1 and 2) and (3 and 4) differ only in the third dimension and because
% PCA is a linear transformation it wasn't able to separate them. However Isomap,
% a nonlinear dimensionality reduction method, on the other hand was able to
% separate all the four clusters.


% Loading the MNIST data
X = zeros(10000, 784);
for dig = 1:10;
    fid=fopen(['data', num2str(dig-1)], 'r');
    for exa = 1:1000;
        temp = fread(fid, [28 28]);
        X((dig-1)*1000+exa, :) = temp(:)';
%         colormap(gray(256));
%         image(temp);
    end
end
fclose(fid);

% P5Q5c)
% As we increase the value of k, the separation gets better and better.
% However, for sufficiently large values of k, the results get messier.

k = 100;
dim = 2;
Xr = isomap(D, dim, k)';
figure; hold on;
plot(Xr(1:400, 1), Xr(1:400, 2), 'rx');
plot(Xr(401:800, 1), Xr(401:800, 2), 'g*');
plot(Xr(801:1200, 1), Xr(801:1200, 2), 'b+');
plot(Xr(1201:1600, 1), Xr(1201:1600, 2), 'ko');
xlabel('x1');
ylabel('x2');
title('SwissRoll - Reduced Data - Isomap with k = 100');
hold off;
```


SwissRoll - Reduced Data - Isomap with k = 4
x2
x1
40
30
20
10
0
-10
-20
-30
-50
-40
-30
-20
-10
0
10
20
30
40



Published with MATLAB® R2015b

Contents


```
% Initialization
close all;
clear all;
clc;
```

Reading and formating for Pima Diabetes Dataset

First row only contains variable count so we don't need that columns from 1st to 2nd last textscan returns a cell so convert it into matrix y is a string so convert it into 0, 1 form.

```
fid = fopen('pima_indians_diabetes.csv', 'r');
textscan(fid, '%f,%f,%f,%f,%f,%f,%f,%f,%s', 1);
raw = textscan(fid, '%f,%f,%f,%f,%f,%f,%f,%f,%s');
X = cell2mat(raw(1:end-1));
y = cell2mat(raw{end})==repmat('pos', size(X, 1), 1);
y = double(y(:, 1));
fclose(fid);

% train/test split for LDA and QDA
XTr = X(1:500, :);
yTr = y(1:500);
XTe = X(501:end, :);
yTe = y(501:end, :);
```

Quadratic discriminant analysis

```
qdaDiscr = fitcdiscr(XTr, yTr,'DiscrimType', 'Quadratic');

[qdaLabelTr, qdaPosteriorTr, qdaProbTr] = predict(qdaDiscr, XTr);
[~, qdaPredTr] = max(qdaProbTr, [], 2);
qdaErrorTr = length(yTr(qdaPredTr==yTr))/length(yTr)*100;
% fprintf('QDA Training Classification Error = %f\n',qdaErrorTr);

[qdaLabelTe, qdaPosteriorTe, qdaProbTe] = predict(qdaDiscr, XTe);
[~, qdaPredTe] = max(qdaProbTe, [], 2);
qdaErrorTe= length(yTe(qdaPredTe==yTe))/length(yTe)*100;
fprintf('QDA Test Classification Error = %f\n\n',qdaErrorTe);
```

```
QDA Test Classification Error = 19.029851
```

Linear discriminant analysis

```
ldaDiscr = fitcdiscr(XTr, yTr, 'DiscrimType', 'DiagLinear');
[ldaLabelTr, ldaPosteriorTr, ldaProbTr] = predict(ldaDiscr, XTr);
[~, ldaPredTr] = max(ldaProbTr, [], 2);
ldaErrorTr = length(yTr(ldaPredTr==yTr))/length(yTr)*100;
% fprintf('LDA Training Classification Error = %f\n',ldaErrorTr);

[ldaLabelTe, ldaPosteriorTe, ldaProbTe] = predict(ldaDiscr, XTe);
[~, ldaPredTe] = max(ldaProbTe, [], 2);
ldaErrorTe= length(yTe(ldaPredTe==yTe))/length(yTe)*100;
fprintf('QDA Test Classification Error = %f\n\n',ldaErrorTe);
```

```
QDA Test Classification Error = 21.641791
```

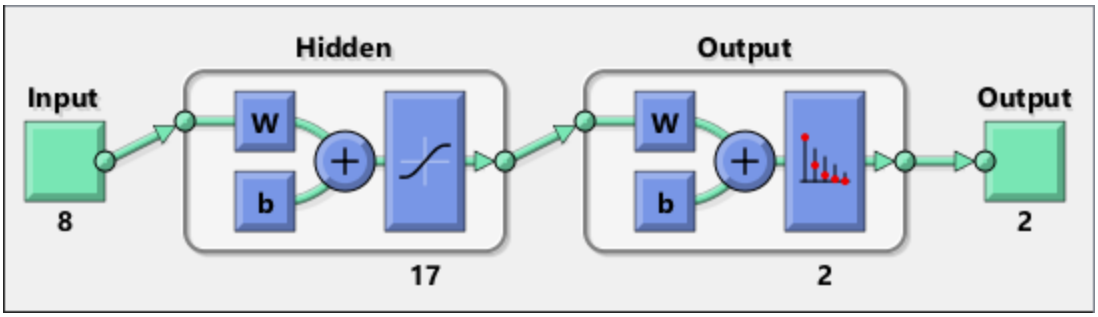
MLP

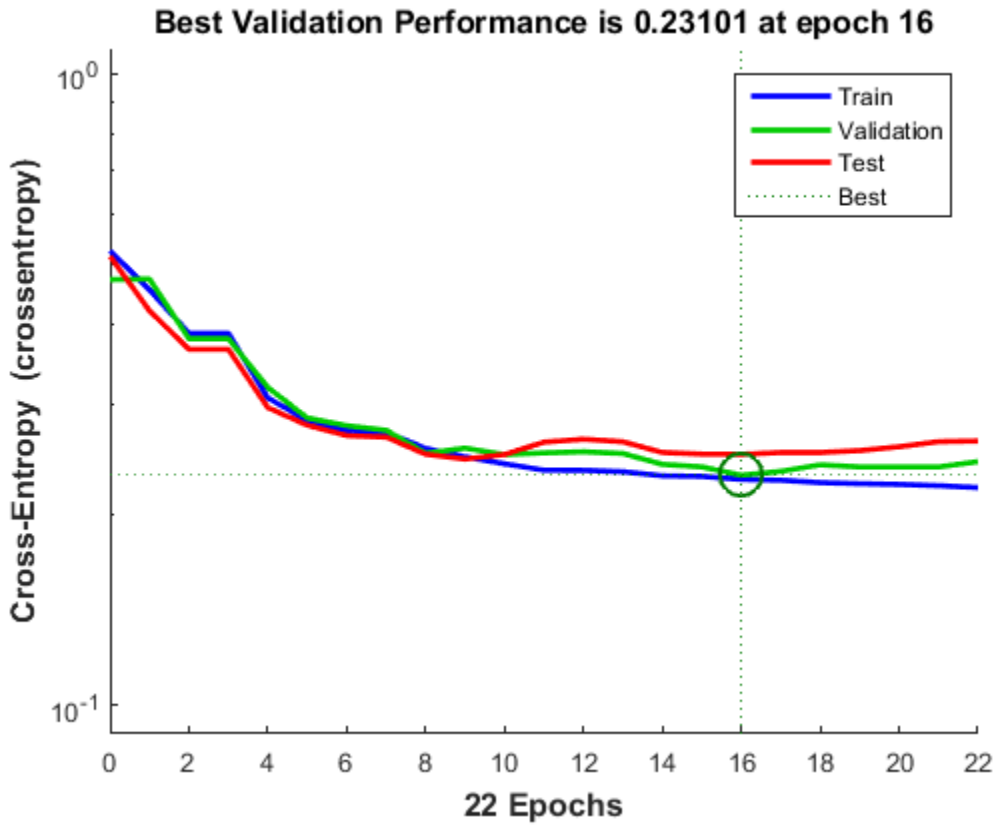
Solve a Pattern Recognition Problem with a Neural Network

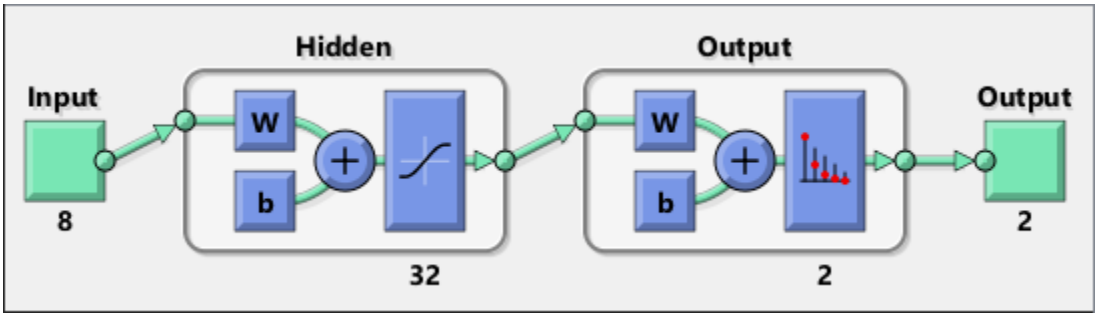
```
% reformat x and y to be indentical to inputs and targets of the Breast
% Cancer dataset
y = vertcat(y',~y');

diabetesInputs = X';
diabetesTargets = y;
x = diabetesInputs;
t = diabetesTargets;

trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a struct for each NN test case with inital values.
mlp = struct('hL', 0, 'hU', 0, 'teErr', 0, 'trainFunc', 'trainscg');

% Create 5 random MLPs and train them
for i =1:5
    mlp(i).hL = randi(50,1,1);
    mlp(i).hU = randi(50,1,1);
    mlp(i).teErr = runMLP(x,t,mlp(i).hL, mlp(i).hU, mlp(i).trainFunc);
end

% find lowest TeErr
```

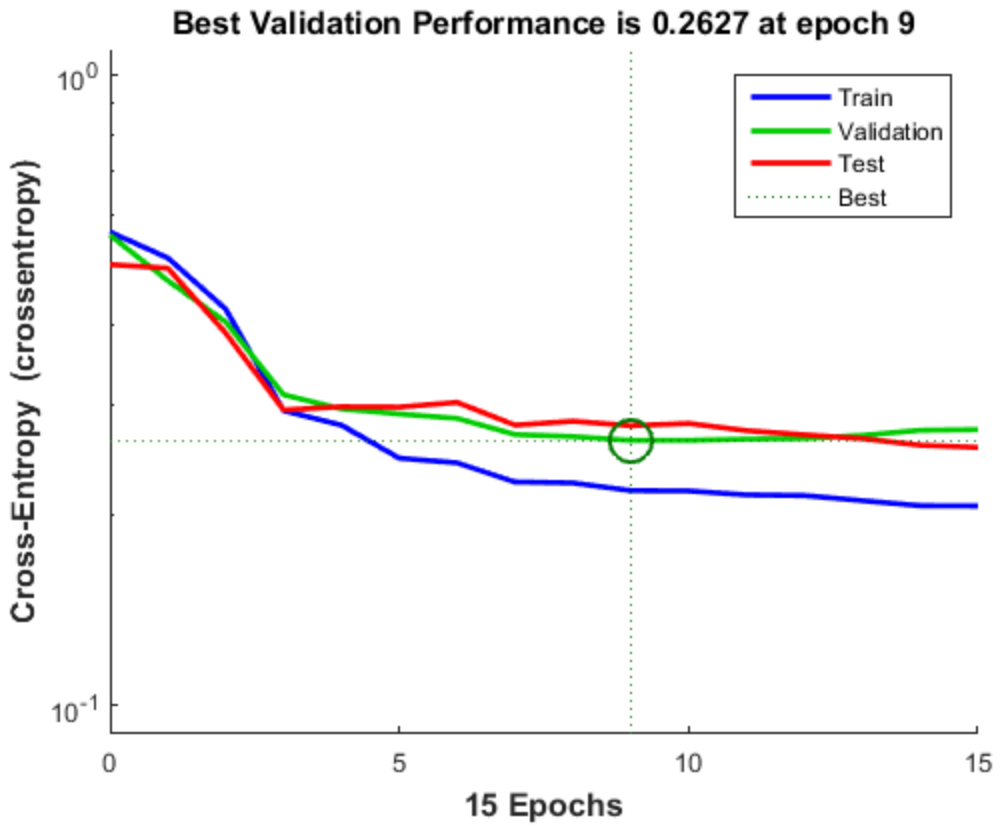
```
teErrMLPArray = arrayfun(@(struct)min(struct.teErr(:)),mlp);
minTeErr = min(teErrMLPArray);
lowestInd = 0;
for i =1:5
    if mlp(i).teErr == minTeErr
        fprintf('The lowest Test Error was with this settings:\n');
        fprintf('hL = %d, hU = %d, teErr = %f\n', mlp(i).hU, mlp(i).hU, mlp(i).teErr);
        lowestInd = i;
    end
end

figure
name = {'LDA';'QDA';'MLP'};
x = [1:3];
y = [ldaErrorTe,qdaErrorTe,minTeErr];
bar(x,y);
title('LDA vs QDA vs MLP')
set(gca,'xticklabel',name);
```

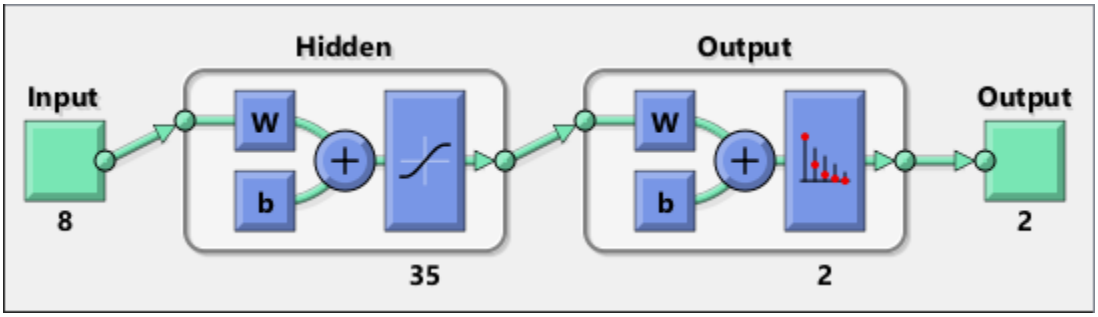
```
MLP with 17 hL has a Test Classification Error = 24.927053
MLP with 32 hL has a Test Classification Error = 27.725642
MLP with 35 hL has a Test Classification Error = 25.995947
MLP with 20 hL has a Test Classification Error = 26.702306
MLP with 26 hL has a Test Classification Error = 20.933196
The lowest Test Error was with this settings:
hL = 12, hU = 12, teErr = 20.933196
```

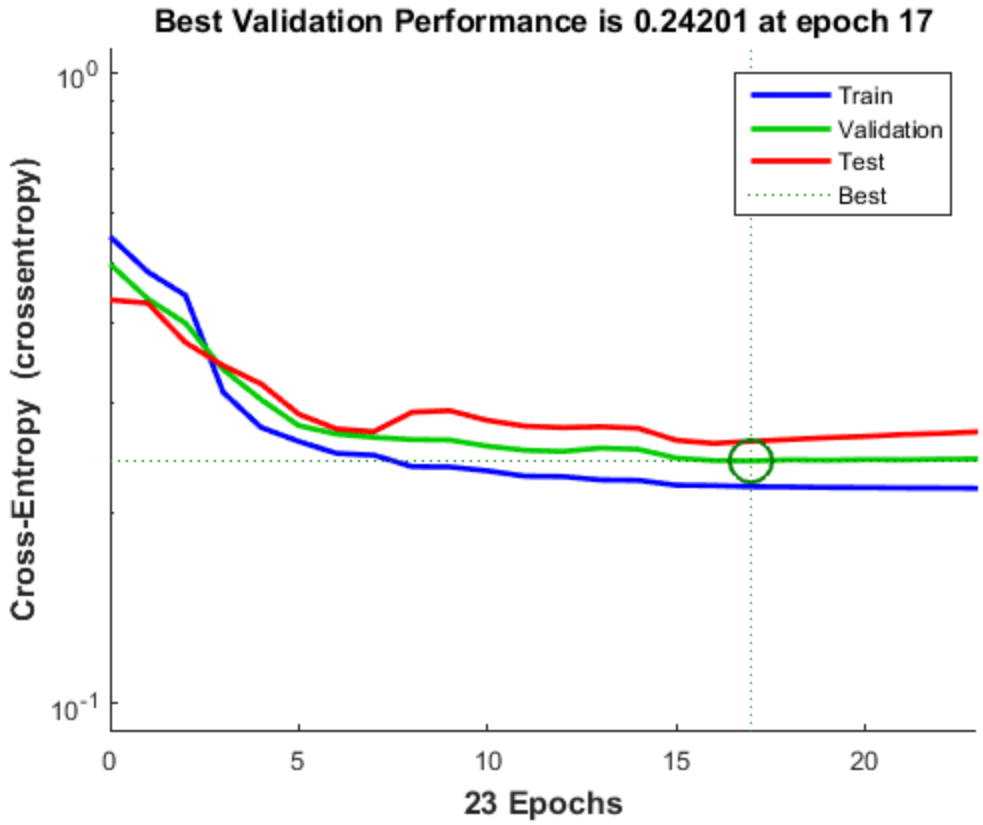

Hidden
Output
Input
W
b
W
b
Output
8
17
2
2

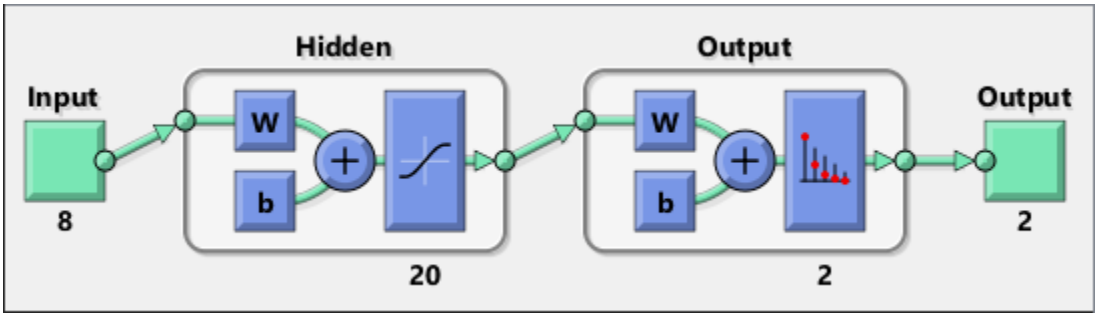

Best Validation Performance is 0.23101 at epoch 16
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
10^0
10^{-1}
0
2
4
6
8
10
12
14
16
18
20
22
22 Epochs

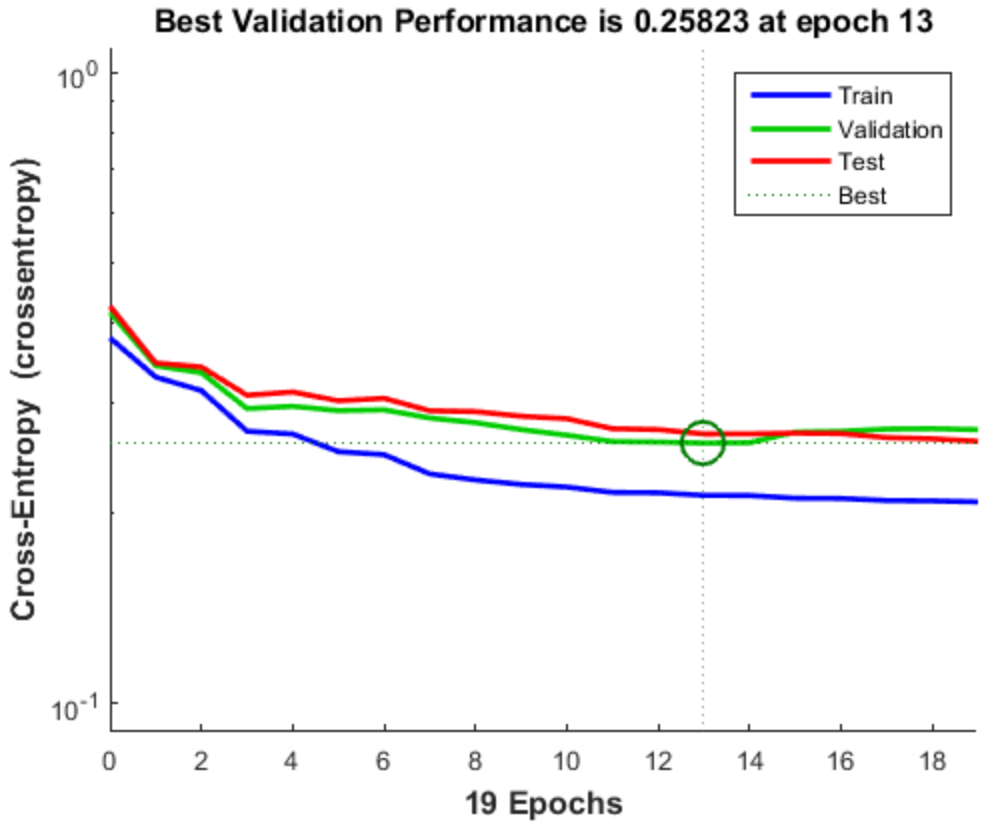

Hidden
Output
Input
W
b
W
b
Output
8
32
2
2

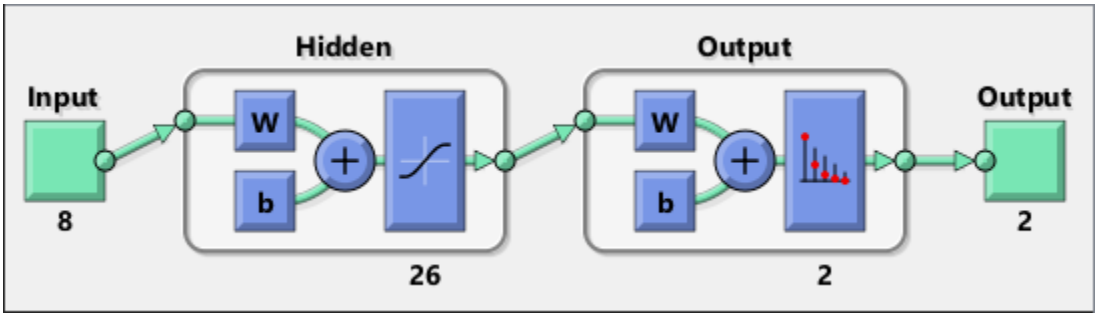

Best Validation Performance is 0.2627 at epoch 9
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
15 Epochs

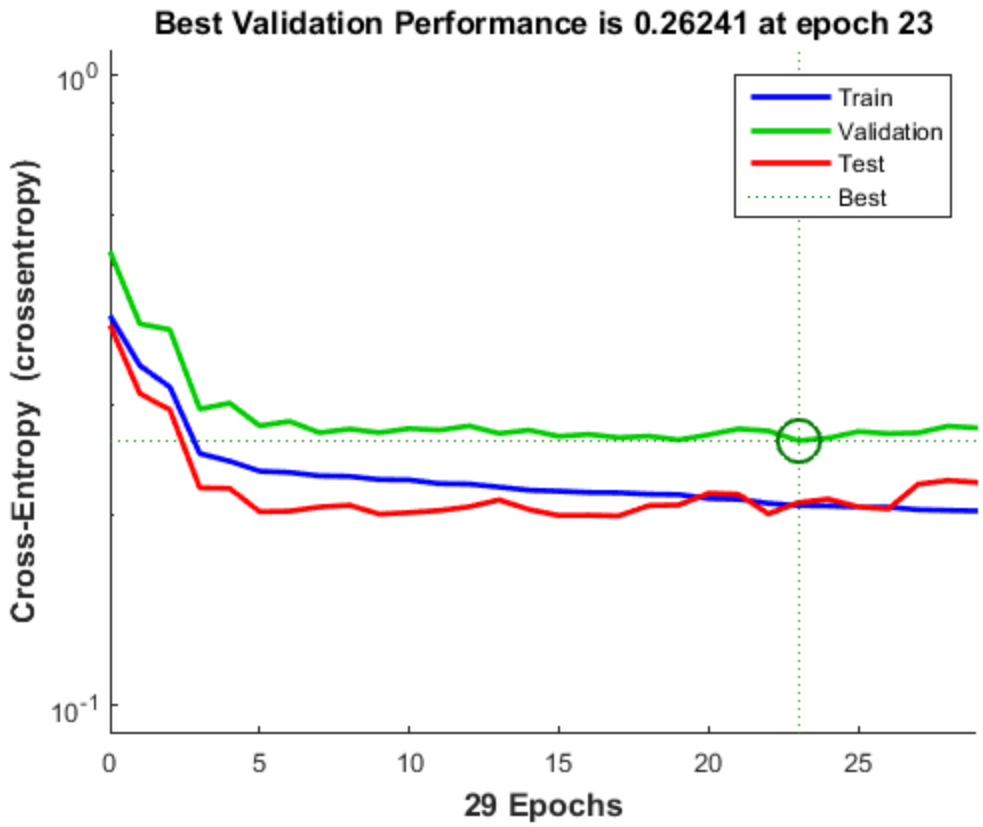

Input
8
Hidden
W
b
35
Output
W
b
2
Output
2

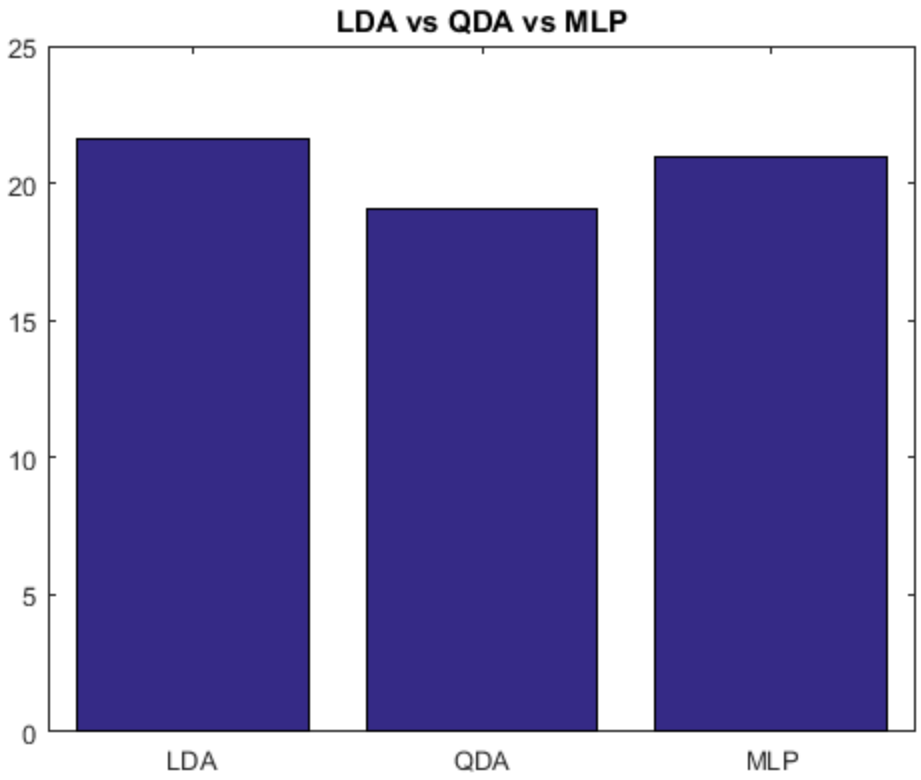

Best Validation Performance is 0.24201 at epoch 17
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
10^0
10^{-1}
0
5
10
15
20
23 Epochs


Input
8
Hidden
W
b
20
Output
W
b
2
Output
2


Best Validation Performance is 0.25823 at epoch 13
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
19 Epochs


Input
8
Hidden
W
b
26
Output
W
b
2
Output
2


Best Validation Performance is 0.26241 at epoch 23
Train
Validation
Test
Best
Cross-Entropy (crossentropy)
10^0
10^{-1}
0
5
10
15
20
25
29 Epochs



Published with MATLAB® R2015b