4.2

Presetting the radius of flat kernel function to be radius(bandwidth) = 2

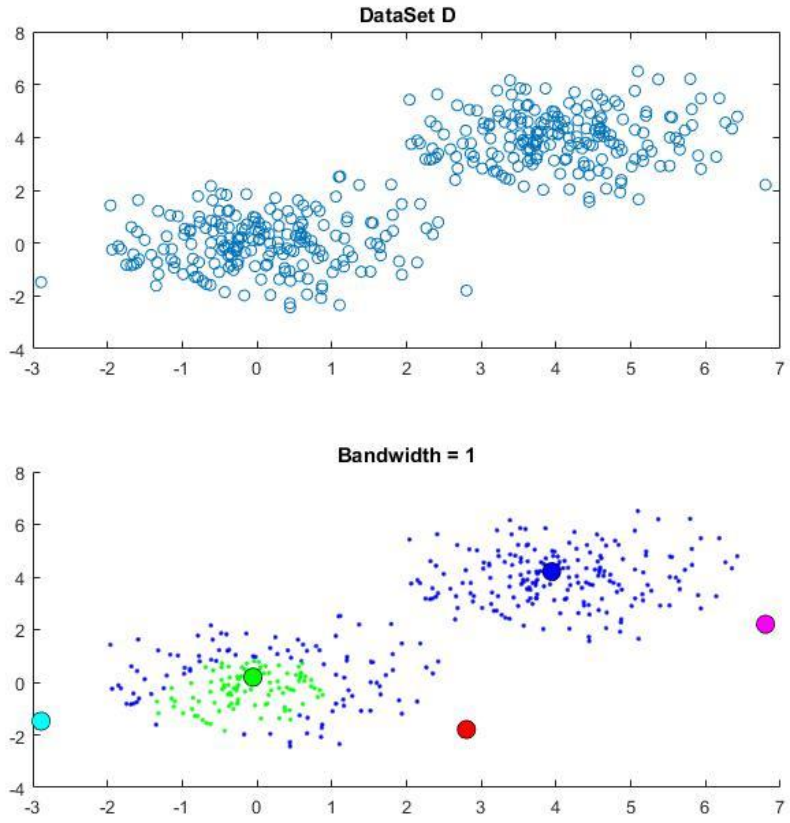
Matlab Code:

```
function [clustCent,data2cluster, cluster2dataCell] = sample-
MeanShiftAlgorithm(dataPts, bandWidth)

%**** Initialize stuff ***
[numDim,numPts] = size(dataPts);
numClust = 0;  %number of output cluster
radius = 2; %radius for flat kernel function
% radius = bandWidth; if specify
bandSq  = radius^2;
initPtInds  = 1:numPts;
stopThresh = 1e-3*radius; %when sampleMean has converged
clustCent = []; %center of clust
%track if a points been seen already
beenVisitedFlag = zeros(1,numPts,'uint8');
%number of points to posibaly use as initilization points
numInitPts = numPts;
%used to resolve conflicts on cluster membership
clusterVotes    = zeros(1,numPts,'uint16');

while numInitPts

    % pick a random seed point
    tempInd = ceil( (numInitPts-1e-6)*rand);
    % use this point as start of sampleMean
    stInd = initPtInds(tempInd);
    % intilize sampleMean to this points location
    sampleMean = dataPts(:,stInd);
    % points that will get added to this cluster
    myMembers  = [];
    % used to resolve conflicts on cluster membership
    thisClusterVotes = zeros(1,numPts,'uint16');

    while 1     %loop untill convergence
        %dist squared from sampleMean to all points still active
        sqDistToAll = sum((repmat(sampleMean,1,numPts) - dataPts).^2);
        inInds = find(sqDistToAll < bandSq); %points within radius
        %add a vote for all the in points belonging to this cluster
        thisClusterVotes(inInds) = thisClusterVotes(inInds)+1;
        presampleMean   = sampleMean; %save the old sampleMean
        sampleMean      = mean(dataPts(:,inInds),2);  %compute the new
sampleMean
        % add any point within radius to the cluster
        myMembers   = [myMembers inInds];
        % mark that these points have been visited
        beenVisitedFlag(myMembers) = 1;
        % if sampleMean doesn't move much stop this cluster
        if norm(sampleMean-presampleMean) < stopThresh
            %check for merge posibilities
            mergeWith = 0;
            for cN = 1:numClust
                %distance from posible new clust max to old clust max
                distToOther = norm(sampleMean-clustCent(:,cN));
                %if its within radius/2 merge new and old
                if distToOther < radius/2
                    mergeWith = cN;
                    break;
```

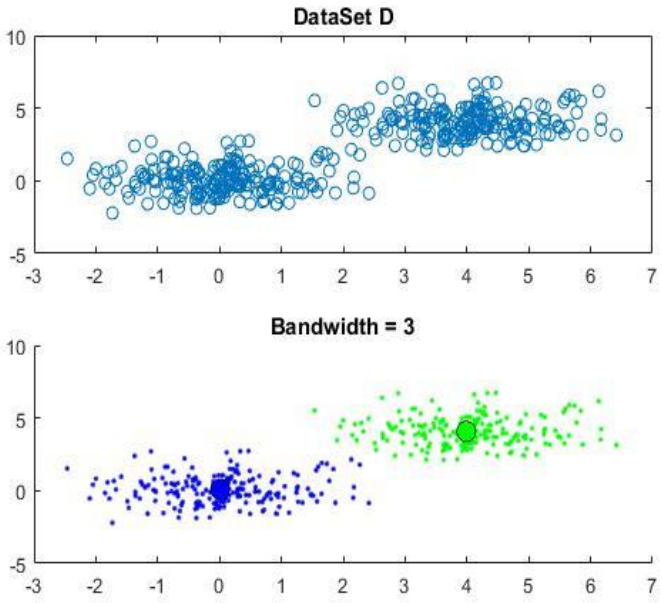
```
            end
            if mergeWith > 0    % something to merge
                % Record the max as the sampleMean of the two merged
                clustCent(:,mergeWith) = 0.5*(sample-
Mean+clustCent(:,mergeWith));
                % Add these votes to the merged cluster
                clusterVotes(mergeWith,:) = clusterVotes(mergeWith,:)
                + thisClusterVotes;
            else    %its a new cluster
                numClust = numClust+1; %increment clusters
                clustCent(:,numClust)  = sampleMean; %record the sam-
pleMean
                % Add these votes to the merged cluster
                clusterVotes(numClust,:)    = thisClusterVotes;
            end
            break;
        end
    end

    % initialize with any of the points not yet visited
    initPtInds = find(beenVisitedFlag == 0);
    numInitPts = length(initPtInds); %number of active points in set

end

% A point belongs to the cluster with the most votes
[~,data2cluster] = max(clusterVotes,[],1);

% Calculate cluster2data cell
if nargout > 2
    cluster2dataCell = cell(numClust,1);
    for cN = 1:numClust
        myMembers = find(data2cluster == cN);
        cluster2dataCell{cN} = myMembers;
    end
end
```

4.3

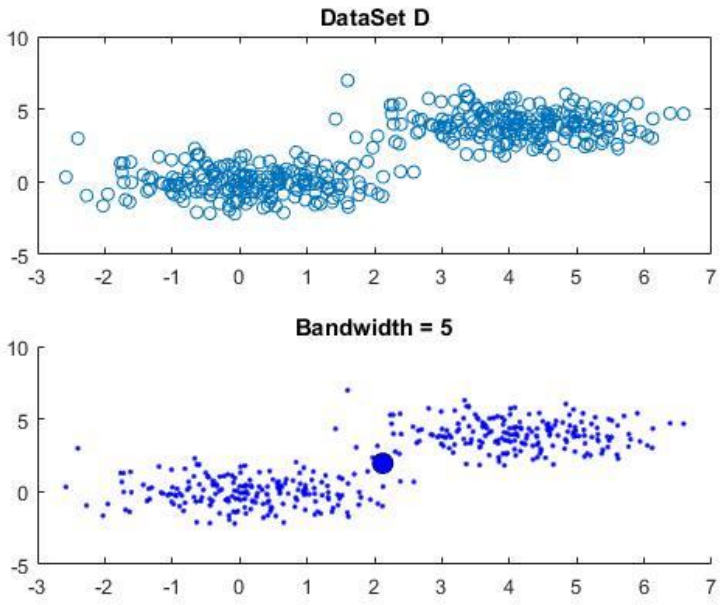
(1). Dataset D, choose radius(bandwidth) value r = 1, 3, 5, we get the result as



When bandwidth = 1, it can be seen that the number of clusters generated is 5, which is not we expected. The issue is that the dataset is consist of two multivariate Gaussian models with different mean value, which tends to be two clusters intuitively, however, the radius of kernel function is too small so that it cannot cover all the data points on the figure, which results in the extra points (especially edging point) will form a new cluster with very few points.

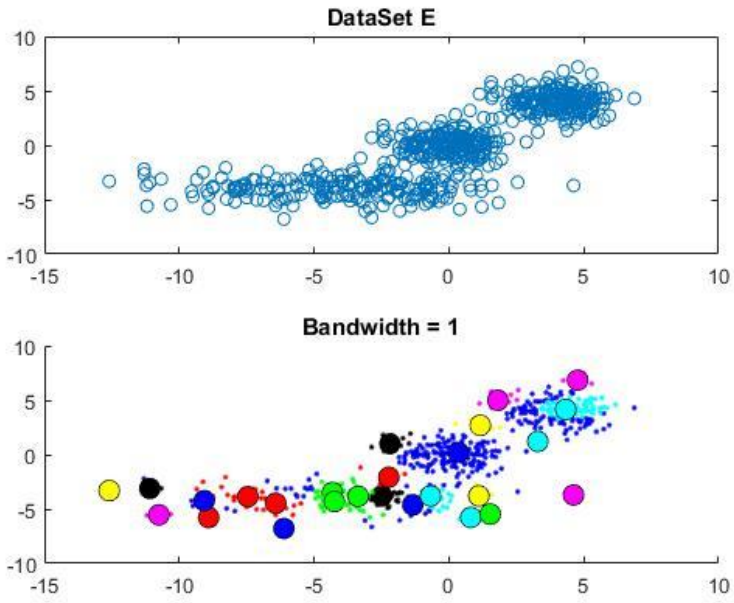


When bandwidth = 3, it can be seen that the number of clusters generated is 2. This model performs well under this bandwidth.

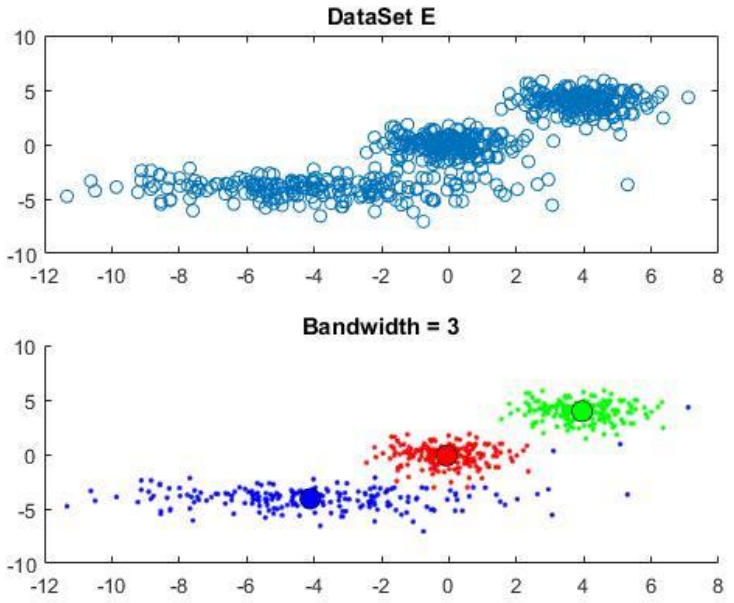


When bandwidth = 5, it can be seen that the number of clusters generated is 1, which is too few in this case. That's because the bandwidth of the kernel function is too large, which makes the initial cluster cover all the data points so there are no points out of range of first cluster and the loop end.

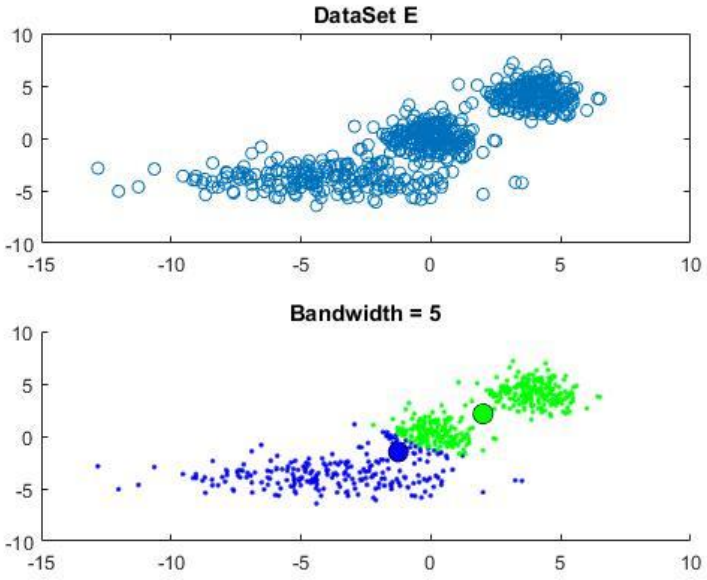
(2). Same as dataset d, now we try to modelling dataset e by the same radius (bandwidth) r = 1, 3, 5



Firstly, we can see the dataset E is consisting mainly from three regions, which expecting to see the number of clusters to be 3. When bandwidth = 1, it can be seen that the model generating a lots of clusters, and each one of them only representing few data points, which is due to the smaller value of bandwidth and it's not we expected.



When bandwidth is equal to three, it can be seen that three clusters have been generated, and each of them have a clear centre which represent the most density part in that cluster. It is a good model to representing this dataset since we expected the clusters numbers to be three intuitively.



When bandwidth is equal to five, it can be seen there are only two clusters in this case, which is not we expected since it does not successfully distinguish the top two clusters, and the reason for that is the bandwidth of kernel function are too wide, same as dataset d.

Attached: mean shift algorithm test code

```
clear
profile on
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(1);

d = transpose(d);
e = transpose(e);
subplot(2,1,1);
plot(e(1,:),e(2,:),'o');
title('DataSet E')
bandWidth = 5

tic
[clustCent,data2pointer,clustMembsCell] = MeanShiftAlgorithm(e,
bandWidth);
toc

numClust = length(clustMembsCell);
subplot(2,1,2);
hold on
cVec = 'bgrcmykbgrcmykbgrcmykbgrcmyk';%, cVec = [cVec cVec];
for k = 1:min(numClust,length(cVec))
    myMembers = clustMembsCell{k};
    myClustCen = clustCent(:,k);
    plot(e(1,myMembers),e(2,myMembers),[cVec(k) '.'])

plot(myClustCen(1),myClustCen(2),'o','MarkerEdgeColor','k','MarkerFaceC
olor',cVec(k), 'MarkerSize',10)
end
title('Bandwidth = 5')
```

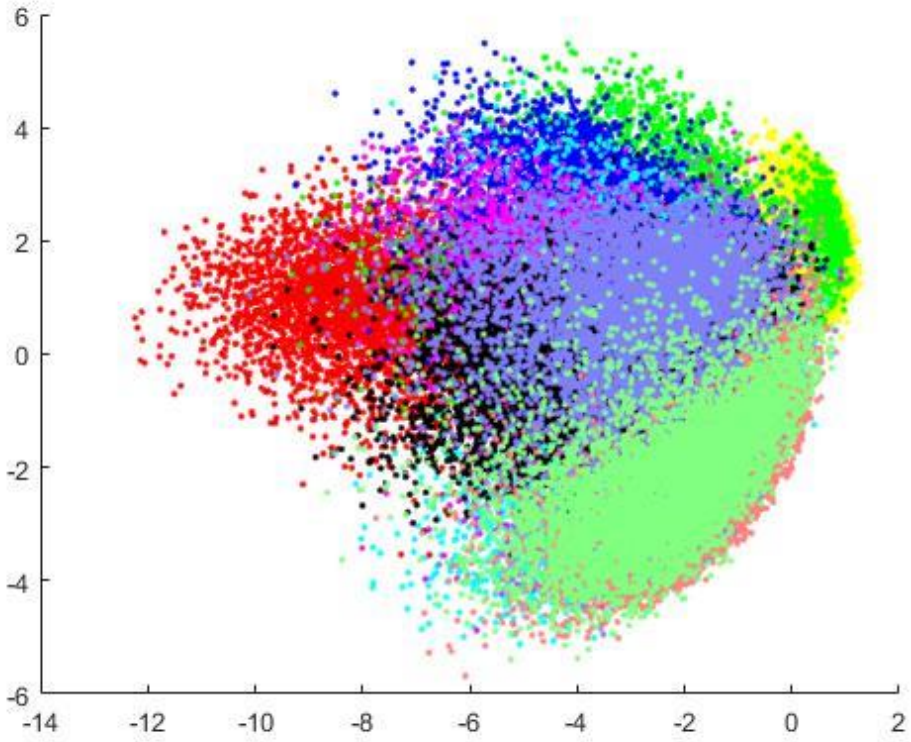
5.1

Principal Component Analysis with reduced dimension d = 2:

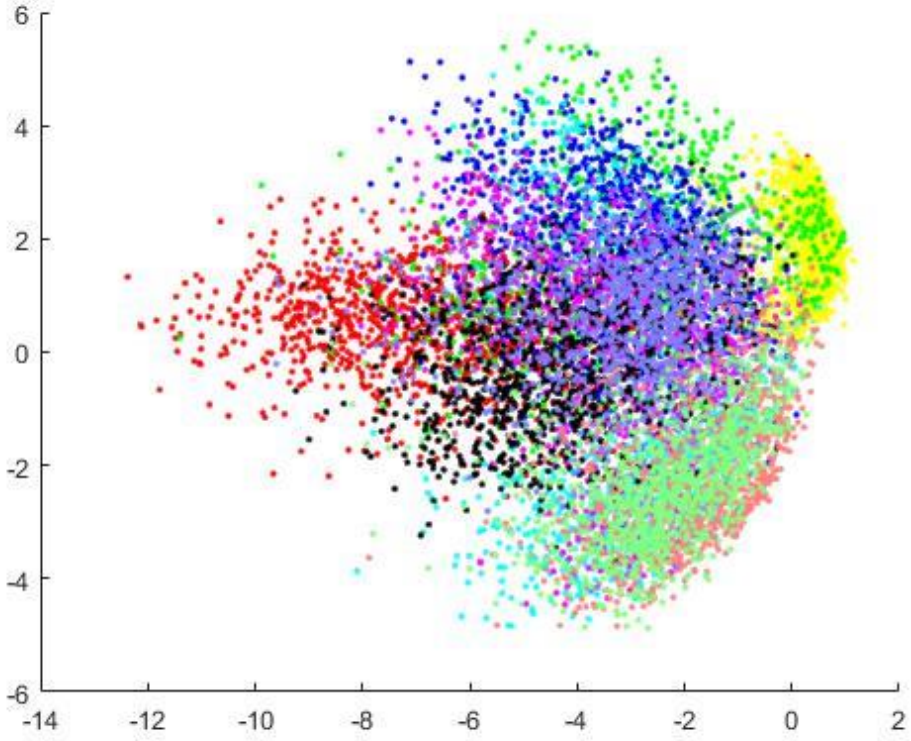
```
function [evector, evalue, reduced_data, var_percent] = Pca(dataSet)
    % Extract the number of C and P of this dataset
    [numC, numP] = size(dataSet);
    % Finding covariance matrix
    covariancematrix=cov(dataSet);
    % Finding eigenvector and eigenvalue
    [evector,evalue] = eig(covariancematrix);
    % Initialising max and second max eigenvector and eigenvalue
    maxEvalue = 0;
    maxIndex = 0;
    nextEvalue = 0;
    sumEigenValue = 0;
    maxEvector = zeros(numP,2);

    % Finding first maximum eigenvalue and store eigenvector
    for i = 1:numP
        if maxEvalue < evalue(i,i)
            maxEvalue = evalue(i,i);
            maxIndex = i;
        end
    end
    for j = 1:numP
        maxEvector(j,1) = evector(j,maxIndex);
    end

    % Finding second maximum eigenvalue and store eigenvector
    for i = 1:numP
        if nextEvalue < evalue(i,i) && evalue(i,i) ~= maxEvalue
            nextEvalue = evalue(i,i);
            maxIndex = i;
        end
        sumEigenValue = sumEigenValue + evalue(i,i);
    end
    for j = 1:numP
        maxEvector(j,2) = evector(j,maxIndex);
    end

    % Outputing the reduced dataset
    reduced_data = dataSet * maxEvector;

    % Generating the proportion of variance
    sumMaxEvalue = maxEvalue + nextEvalue;
    var_percent = sumMaxEvalue/sumEigenValue;

end
```

5.2

(1). The reduced data plotted in a two-dimension as shows (mnist_train)



Testing the PCA with mnist_test, we can get reduced data plotted as below.

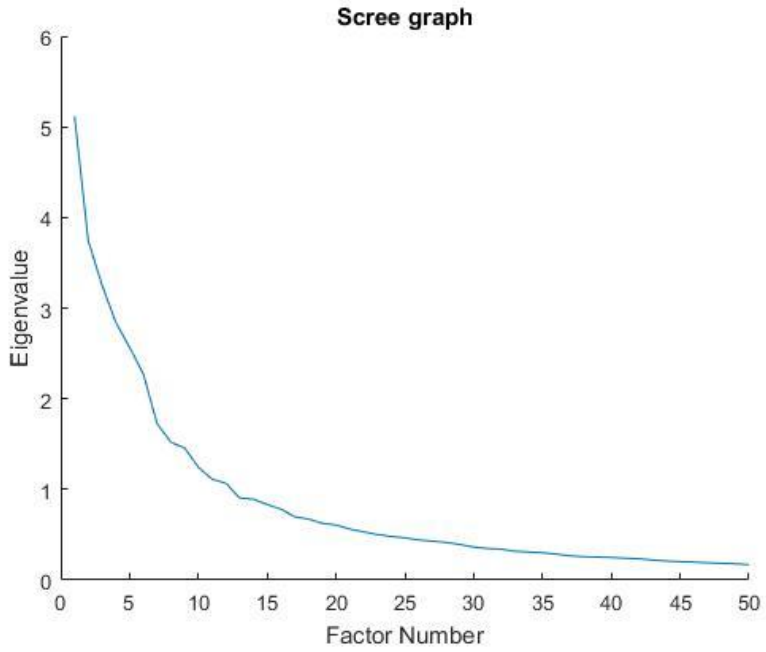


(2). Based on the output for the percentage of eigenvalue, the proportion of variance is calculated as 0.1680 (mnist_train).

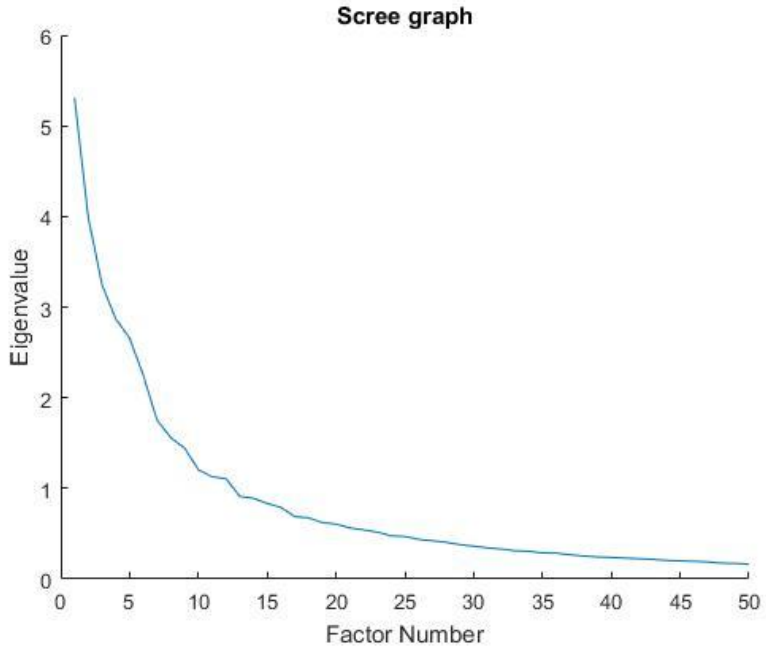
Testing under mnist_test, the proportion of variance is 0.1759.

(3). Scree graph: Since after 50 factors, the eigenvalue will significantly tends to be 0, so in here it only shows the first maximum eigenvalue to the fiftieth maximum one.

Mnist_train scree graph:



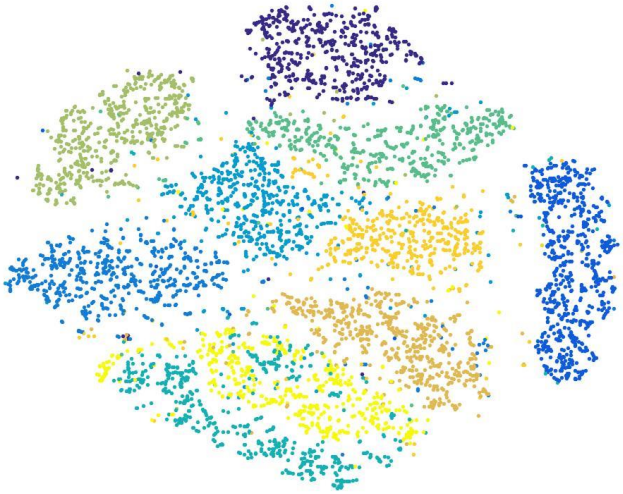
Mnist_test scree graph:



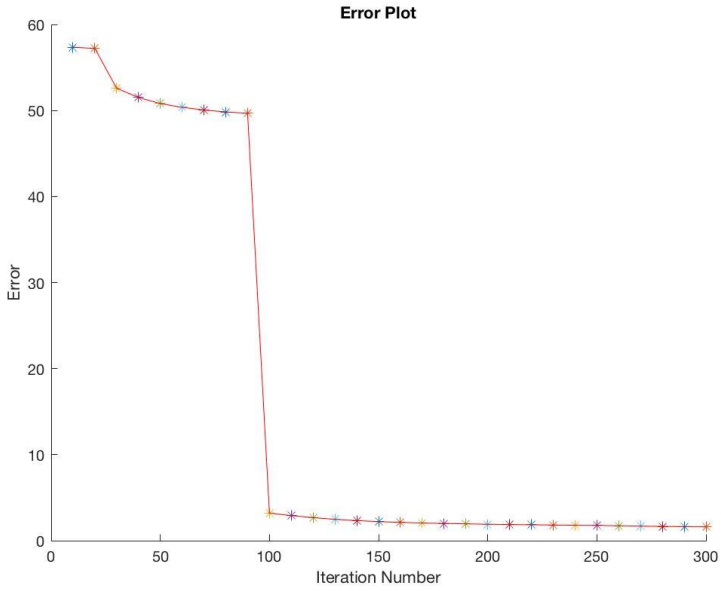
5.6

Running t-SNE algorithm with provided testing code, we can get the plot like below.

The following diagram shows that the data of mnist_train set be training using t-SNE algorithm when iteration i = 300.

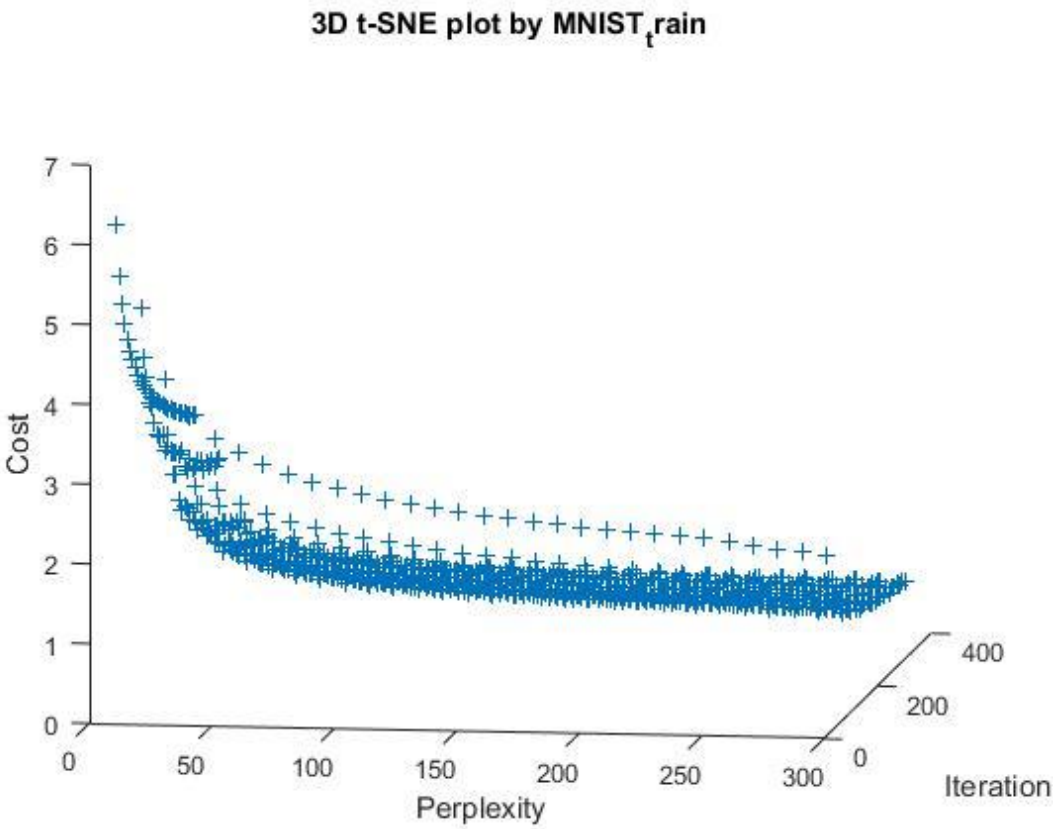


The below plot shows the error at each iteration up to 300 iterations in steps of 10.

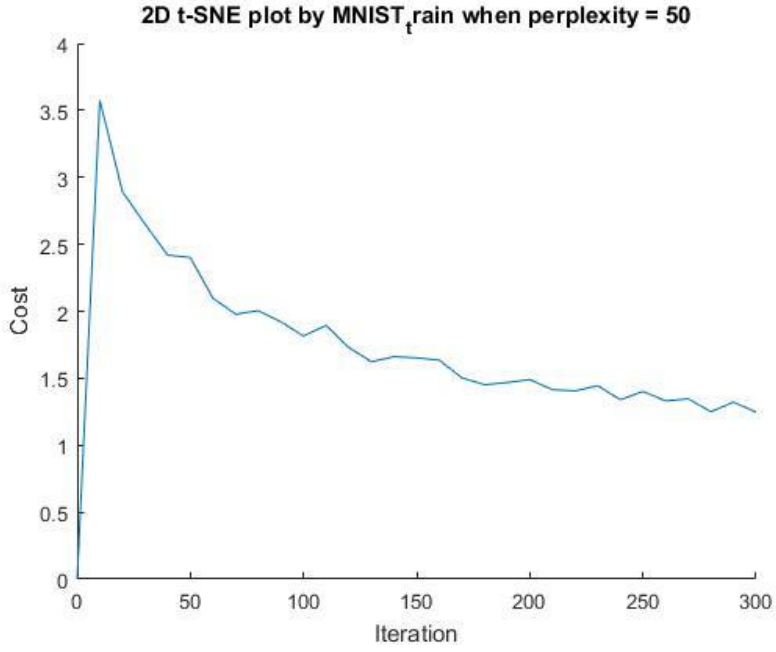


5.8

The 3D plot generated by iteration, perplexity and cost as follow shows.



From above diagram, it can be seen that cost will decrease as the perplexity increasing, however, higher perplexity will cause significant increasing of computation complexity. To balance that, choosing the point where the gradient of cost tends to approaching zero, where the perplexity is 50 as diagram shows. Therefore, we choose 50 as the perplexity value and the following diagram shows the cost vs iteration under perplexity = 50.



6.3

Neutral Network Training under Matlab NN toolbox. (Dataset: MNIST_train)

(1). Choose Input, Hidden and Output layer.

Input nodes: 784 (Fixed by data)

Output nodes: Since the problem is a classification problem with 10 different classes, so we set the number to be 10 represent 10 different output types.

Hidden layers: According from Marsland(2014), two hidden layers is the most that you every need for normal MLP learning, in fact one hidden layer with lots of hidden nodes is sufficient. In here, we choose the number of hidden layers to be one for simplify complexity and time saving.

Hidden nodes: Since we only choose one hidden layers, so we need to set more hidden nodes to making the training accurate. Therefore, we set the hidden nodes to be 12.

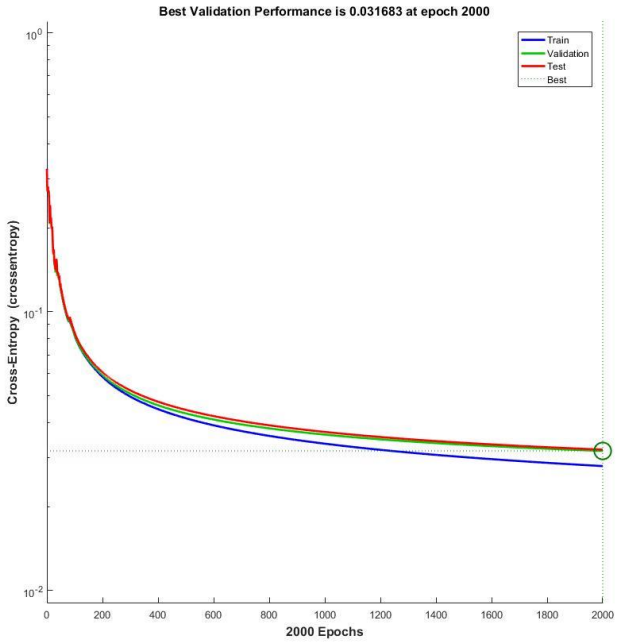
(2). To ensuring the algorithm reliability, we set the maximum iteration of training to be 2000.

(3). Data splitting

To be algorithm reliable, we are using 70% of data for data training, 15% for data validation and 15% for data testing. Using a large proportion of data for training is for ensuring the algorithm accuracy and reliable.

(4). Performance.

To represent data error between data output and target output, we are using cross-entropy to represent the sum-of-square error function. Based on the following diagram, the minimum error of this algorithm, which represents by the best validation performance is 3.168% and still slowly decreasing. If we continue running for some time, this error will be increase again due to overfitting and that's the threshold time when the algorithm stops running. The performance of training set data won't be able to affect too much of the validation and testing data after the error goes below 4%, which is causing by the amount of training data (70%) cannot fully cover the rest of data(30%).



MLP Testing Code

```
load('mnist_train.mat');
labels = [];
for i = 1:10
    labels = [labels, train_labels == i];
end
x = train_X';
t = labels';
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'traingd';  % Scaled conjugate gradient backpropagation.
% Create a Pattern Recognition Network
hiddenLayerSize = 12;
net = patternnet(hiddenLayerSize, trainFcn);
net.trainParam.lr = 0.5;
% Setup the maximum ieration loops
net.trainParam.epochs = 2000;
% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% View the Network
view(net)
```