Coursework 1

Task 1 - Part 1 – PCA Analysis on the Wine Dataset

Answers 1,2,3

Source File: process_wine_data_PCA.m

Code

```
% PCA Analysis of given Wine data
%Load data from text file to Matalb
wine=readtable('wine.data.txt');
% Name all the columns
wine.Properties.VariableNames{1} = 'Class';
wine.Properties.VariableNames{2} = 'Alcohol';
wine.Properties.VariableNames{3} = 'MalicAcid';
wine.Properties.VariableNames{4} = 'Ash';
wine.Properties.VariableNames{5} = 'AlcalOfAsh';
wine.Properties.VariableNames{6} = 'Magnesium';
wine.Properties.VariableNames{7} = 'TotalPhenol';
wine.Properties.VariableNames{8} = 'Flavanoids';
wine.Properties.VariableNames{9} = 'NonFlavanoidPhenols';
wine.Properties.VariableNames{10} = 'Proanthocyanins';
wine.Properties.VariableNames{11} = 'ColorIntensity';
wine.Properties.VariableNames{12} = 'Hue';
wine.Properties.VariableNames{13} = 'OD280_OD315_dw';
wine.Properties.VariableNames{14} = 'Proline';
%Copy Class variable as Labels
Labels=table2array(wine(:,1));
% Remove Class column
wine=removevars(wine,'Class');
% Convert table data to matrix
wine_data=table2array(wine);
% Find mean of matrix
wine_data_mean = mean(wine_data);
% create pre-process data by substracting mean
prep_wine_data=wine_data-wine_data_mean;
% find standard deviation of wine data
wine_data_std=std(wine_data);
% normalized wine data (data-mean)/std
norm_wine_data=prep_wine_data./wine_data_std;
% perform PCA using NETLAB function
[pcvals,pcvecs]=pca(norm_wine_data);
% Create projdata using formula
projdata=norm_wine_data*pcvecs(:,1:2);
% Open Figure 1
figure(1)
hold on;
% Create plot for label 1
h1=plot(projdata(Labels==1,1),projdata(Labels==1,2),'rx','MarkerSize',6);
% Create plot for label 2
h2=plot(projdata(Labels==2,1),projdata(Labels==2,2),'bo','MarkerSize',6);
% Create plot for label 3
h3=plot(projdata(Labels==3,1),projdata(Labels==3,2),'kd','MarkerSize',6);
% Labeling
set(gca,'Box','on');
legend('Class 1','Class 2','Class 3');
xlabel('PC1');
```

```
ylabel('PC2');
save wine_pca.mat
```

PCA Plot

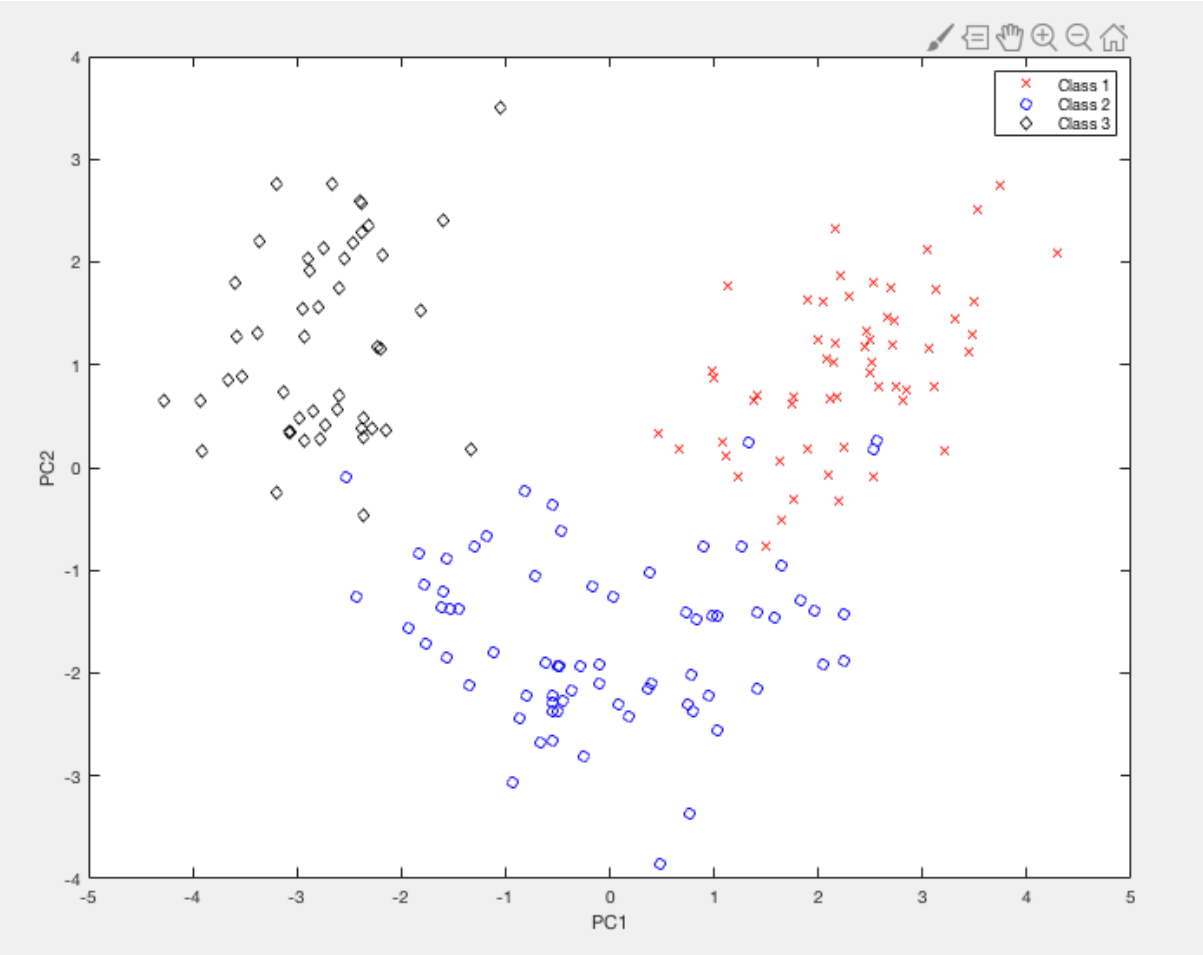


Figure 1

Task 1 - Part 2 – K-Means Clustering

Answer 4

Source File: wine_data_Kmeans.m

```
% K-means clustering on given wine data.
%Load data from text file to Matalb
wine=readtable('wine.data.txt');
% Name all the columns
wine.Properties.VariableNames{1} = 'Class';
wine.Properties.VariableNames{2} = 'Alcohol';
wine.Properties.VariableNames{3} = 'MalicAcid';
wine.Properties.VariableNames{4} = 'Ash';
wine.Properties.VariableNames{5} = 'AlcalOfAsh';
wine.Properties.VariableNames{6} = 'Magnesium';
wine.Properties.VariableNames{7} = 'TotalPhenol';
wine.Properties.VariableNames{8} = 'Flavanoids';
wine.Properties.VariableNames{9} = 'NonFlavanoidPhenols';
wine.Properties.VariableNames{10} = 'Proanthocyanins';
wine.Properties.VariableNames{11} = 'ColorIntensity';
wine.Properties.VariableNames{12} = 'Hue';
```

```
wine.Properties.VariableNames{13} = 'OD280_OD315_dw';
wine.Properties.VariableNames{14} = 'Proline';
%Copy Class variable as Labels
Labels=table2array(wine(:,1));
% Remove Class column
wine=removevars(wine,'Class');
% Convert table data to matrix
wine_data=table2array(wine);
% Find mean of matrix
wine_data_mean = mean(wine_data);
% create pre-process data by substracting mean
prep_wine_data=wine_data-wine_data_mean;
% find standard deviation of wine data
wine_data_std=std(wine_data);
% normalized wine data (data-mean)/std
norm_wine_data=prep_wine_data./wine_data_std;
% start k-Means clustering process, load normalized data
% Set parameters and call K means for K=3
data=norm_wine_data;
rand('state',0);
ndata = size(data, 1);
ncentres = 3; % 3 codevectors
perm = randperm(ndata);
perm = perm(1:ncentres);
centres = data(perm, :);
options = foptions;
options(1)  = 1;        % Prints out error values.
options(14) = 10;       % Number of iterations.
% Train the centres from the data
disp('Perform K means using 3 code vectors');
[centres, options, post] = kmeans(centres, data, options);
[one_value, membership] = max(post,[],2);
% repeat the same process for K=5
data=norm_wine_data;
rand('state',0);
ndata = size(data, 1);
ncentres = 5; % 5 codevectors
perm = randperm(ndata);
perm = perm(1:ncentres);
centres = data(perm, :);
options = foptions;
options(1)  = 1;        % Prints out error values.
options(14) = 15;       % Number of iterations.
% Train the centres from the data
disp('-----------------------------------');
disp('Perform K means using 5 code vectors');
[centres, options, post] = kmeans(centres, data, options);
[one_value, membership] = max(post,[],2);
```

Answer 5

Error Reporting

Output – Error rates for 3 and 5 code vectors respectively

```
>> wine_data_Kmeans
Perform K means using 3 code vectors
Cycle    1  Error 2026.472152
Cycle    2  Error 1365.889924
Cycle    3  Error 1292.415572
```

```
Cycle   4  Error 1280.309132
Cycle   5  Error 1275.767986
Cycle   6  Error 1275.258667
Cycle   7  Error 1275.258667
-----------------------------------
Perform K means using 5 code vectors
Cycle   1  Error 1825.619770
Cycle   2  Error 1205.963905
Cycle   3  Error 1151.456348
Cycle   4  Error 1142.328109
Cycle   5  Error 1134.221371
Cycle   6  Error 1132.003226
Cycle   7  Error 1131.673765
Cycle   8  Error 1130.183915
Cycle   9  Error 1128.354557
Cycle  10  Error 1127.269461
Cycle  11  Error 1125.558766
Cycle  12  Error 1124.267073
Cycle  13  Error 1123.039506
Cycle  14  Error 1123.039506
```

As seen from above output, quantization error is less in case of 5 codevectors (centroids), i.e., sum of squared distance of centroid from data point of their respective cluster data points is the least. Hence, 5 is more appropriate number of code vectors we should choose. However, it may also be observed that the more the number of codevectors/centroid, the lesser quantization error will be given by k-means algorithm. Though, one must agree on a K value which gives clusters that can be interpreted.
Here, since there are 13 features, we cannot effectively conclude which K value is better unless we view projections of the results of k-means on Principal Components feature space.

Answer 6

Source code: wine_data_PCA_Kmeans.m

Code

```
% K-means cluster analysis of given wine data.
%Load data from text file to Matalb
wine=readtable('wine.data.txt');
% Name all the columns
wine.Properties.VariableNames{1} = 'Class';
wine.Properties.VariableNames{2} = 'Alcohol';
wine.Properties.VariableNames{3} = 'MalicAcid';
wine.Properties.VariableNames{4} = 'Ash';
wine.Properties.VariableNames{5} = 'AlcalOfAsh';
wine.Properties.VariableNames{6} = 'Magnesium';
wine.Properties.VariableNames{7} = 'TotalPhenol';
wine.Properties.VariableNames{8} = 'Flavanoids';
wine.Properties.VariableNames{9} = 'NonFlavanoidPhenols';
wine.Properties.VariableNames{10} = 'Proanthocyanins';
wine.Properties.VariableNames{11} = 'ColorIntensity';
wine.Properties.VariableNames{12} = 'Hue';
wine.Properties.VariableNames{13} = 'OD280_OD315_dw';
```

```
wine.Properties.VariableNames{14} = 'Proline';
%Copy Class variable as Labels
Labels=table2array(wine(:,1));
% Remove Class column
wine=removevars(wine,'Class');
% Convert table data to matrix
wine_data=table2array(wine);
% Find mean of matrix
wine_data_mean = mean(wine_data);
% create pre-process data by substracting mean
prep_wine_data=wine_data-wine_data_mean;
% find standard deviation of wine data
wine_data_std=std(wine_data);
% normalized wine data (data-mean)/std
norm_wine_data=prep_wine_data./wine_data_std;
%-------------------------------------------------------------------------%
% start k-Means clustering process for K=3, load normalized data
data=norm_wine_data;
rand('state',0);
ndata = size(data, 1);
ncentres = 3; % 3 codevectors
perm = randperm(ndata);
perm = perm(1:ncentres);
centres = data(perm, :);
options = foptions;
options(1)  = 1;        % Prints out error values.
options(14) = 10;       % Number of iterations.
% Train the centres from the data
disp('-----------------------------------');
disp('Perform K means using 5 code vectors');
[centres, options, post] = kmeans(centres, data, options);
[one_value, membership] = max(post,[],2);
%Perform PCA to find pcvecs
[pcvals,pcvecs]=pca(norm_wine_data);
% Create projdata using formula
projdata=norm_wine_data*pcvecs(:,1:2);
% Open Figure 11
figure(11)
hold on;
% Create plot for cluster 1
h1=plot(projdata(membership==1,1),projdata(membership==1,2),'r.','MarkerSize',12);
% Create plot for cluster 2
h2=plot(projdata(membership==2,1),projdata(membership==2,2),'b.','MarkerSize',12);
% Create plot for cluster 3
h3=plot(projdata(membership==3,1),projdata(membership==3,2),'g.','MarkerSize',12);

% Labeling
set(gca,'Box','on');
xlabel('PC1');
ylabel('PC2');
% Project Kmeans centers on PCA Plot (3 Clusters)
projcentres=centres*pcvecs(:,1:2);
hc3=plot(projcentres(:,1),projcentres(:,2),'k.','LineWidth',2,'MarkerSize',18);
legend('Cluster1','Cluster2','Cluster3','CV');
% --- end K means PC projection for K=3 ---%
%-------------------------------------------------------------------------%
% start k-Means clustering process for K=5, load normalized data
data=norm_wine_data;
rand('state',0);
ndata = size(data, 1);
```

```
ncentres = 5; % 5 codevectors
perm = randperm(ndata);
perm = perm(1:ncentres);
centres = data(perm, :);
options = foptions;
options(1)  = 1;        % Prints out error values.
options(14) = 15;       % Number of iterations.
% Train the centres from the data
disp('-------------------------------------');
disp('Perform K means using 5 code vectors');
[centres, options, post] = kmeans(centres, data, options);
[one_value, membership] = max(post,[],2);
%Perform PCA to find pcvecs
[pcvals,pcvecs]=pca(norm_wine_data);
% Create projdata using formula
projdata=norm_wine_data*pcvecs(:,1:2);
% Open Figure 12
figure(12)
hold on;
% Create plot for cluster 1
h4=plot(projdata(membership==1,1),projdata(membership==1,2),'r.','MarkerSize',12);
% Create plot for cluster 2
h5=plot(projdata(membership==2,1),projdata(membership==2,2),'b.','MarkerSize',12);
% Create plot for cluster 3
h6=plot(projdata(membership==3,1),projdata(membership==3,2),'g.','MarkerSize',12);
% Create plot for cluster 4
h7=plot(projdata(membership==4,1),projdata(membership==4,2),'c.','MarkerSize',12);
% Create plot for cluster 5
h8=plot(projdata(membership==5,1),projdata(membership==5,2),'m.','MarkerSize',12);
% Labeling
set(gca,'Box','on');
xlabel('PC1');
ylabel('PC2');
% Project Kmeans centers on PCA Plot (5 Clusters)
projcentres=centres*pcvecs(:,1:2);
hc5=plot(projcentres(:,1),projcentres(:,2),'k.','LineWidth',2,'MarkerSize',18);
legend('Cluster1','Cluster2','Cluster3','Cluster4','Cluster5','CV');
% --- end K means PC projection for K=5 ---%
```

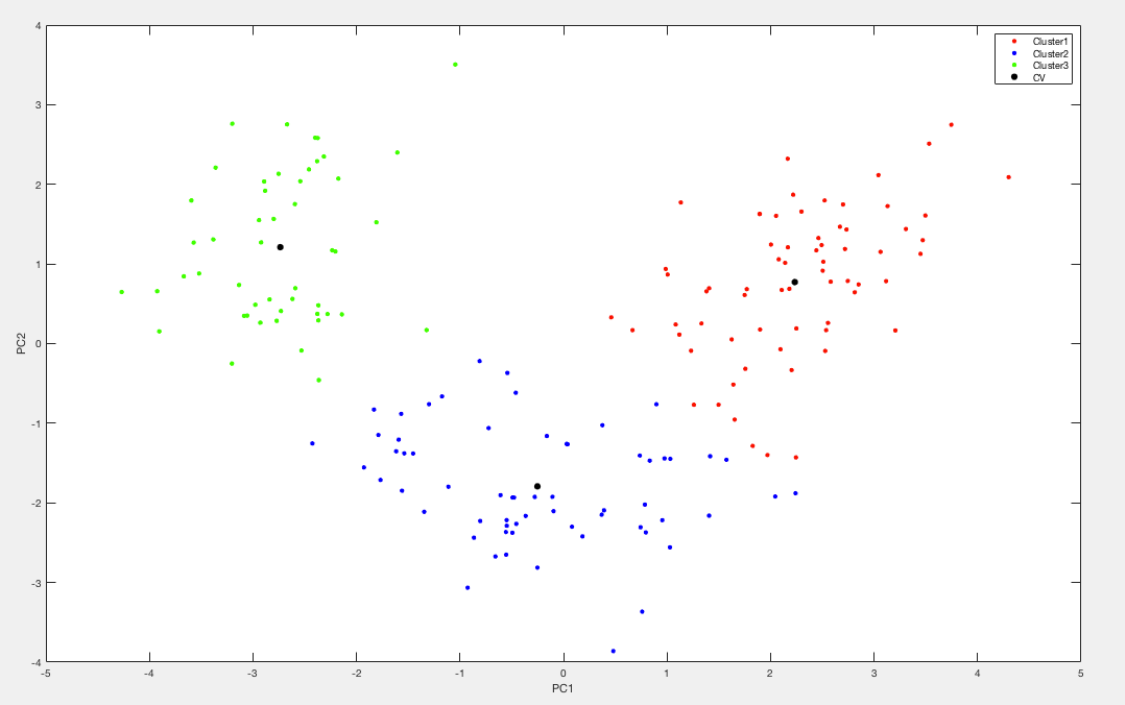


Figure 11 – Projection of data points and code vectors based on k-means, K=3

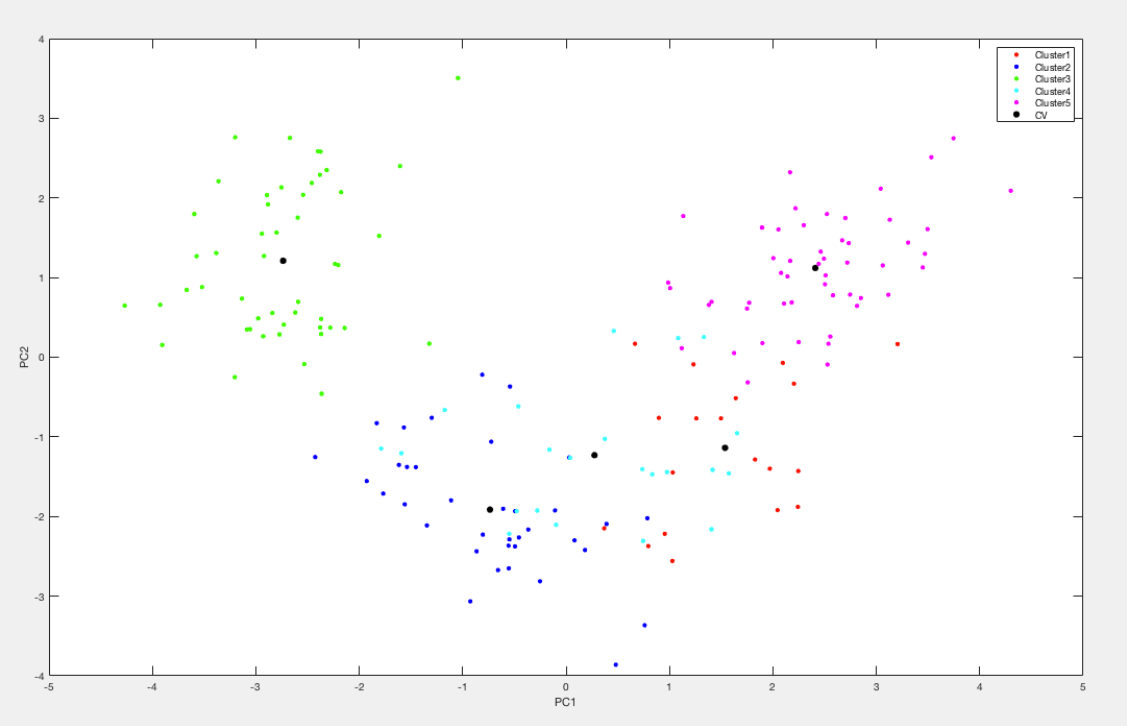


Figure 12 – Projection of data points and code vectors based on k-means, K=5

Interpretation of above figures 11,12 - After executing the K-means algorithm for K values 3 and 5 respectively, we get 3 and 5 cluster memberships as well as their centroid/center points/code-vectors (CV). PCA gives us the eigen vector to transform and view original data in the PC coordinate system. Using these eigen vectors we can project the data points and their centroids on PC1 PC2.

Interestingly, K=3 clustering looks distinct and clear. K=5 gives a slightly confusing view.

Task 1 - Part 3 – Critical Analysis and Summary

The initial plot shown in Figure1 represents all samples of 3 classes of wine data, each belonging to different cultivators. Wine samples of these three cultivators are projected on PC1 and PC2 coordinates after performing Principal component analysis. They indicate that there is a distinction in the prominent features (PC1, PC2), indicating that each cultivator has different chemical composition/features.
Performing k-means clustering on the wine data gives a better idea/further categorization of similar types of wine. For K=3, k-means clustering resembles the PCA projection of given cultivators. Even though error rate achieved in K=5 is lower compared to K=3, K=3 projection appears to be clearer.
Comparing Figure 1 and Figure 11, it can be asserted that few of the given samples of class 2 cultivator might actually belong to class 1 and 3.

Task 2 - Perform image compression using PCA to a fishing boat image

Answers 1,2,3

File: ic_using_pca.m

```
% Load image into matrix - immat
immat=imread('boat.512.tiff');
% Convert back to image - not required
% imagesc(uint8(immat))
% colormap(gray)
% Convert uint8 to double
immat=double(immat);
% Find mean of img data (immat)
mean_immat=mean(immat);
% Find standard deviation of img data
std_immat=std(immat);
% Normalize data
x=(immat-mean_immat);%./std_immat;
% Perform PCA
[pcvals,pcvecs]=pca(x);
% project and recover with different no. of PC components
for n = 1:512
  if n==4 | n==10 | n==15
    recoverImageMatBy4 = proj_and_rec(x,pcvecs,n);
    % add mean and Display Recovered image
    recoverImageMatBy4=recoverImageMatBy4+mean_immat;
    figure(n)
    %imshow(int8(recoverImageMatBy4));
    imagesc(uint8(recoverImageMatBy4));
    colormap(gray);
  end
end
```

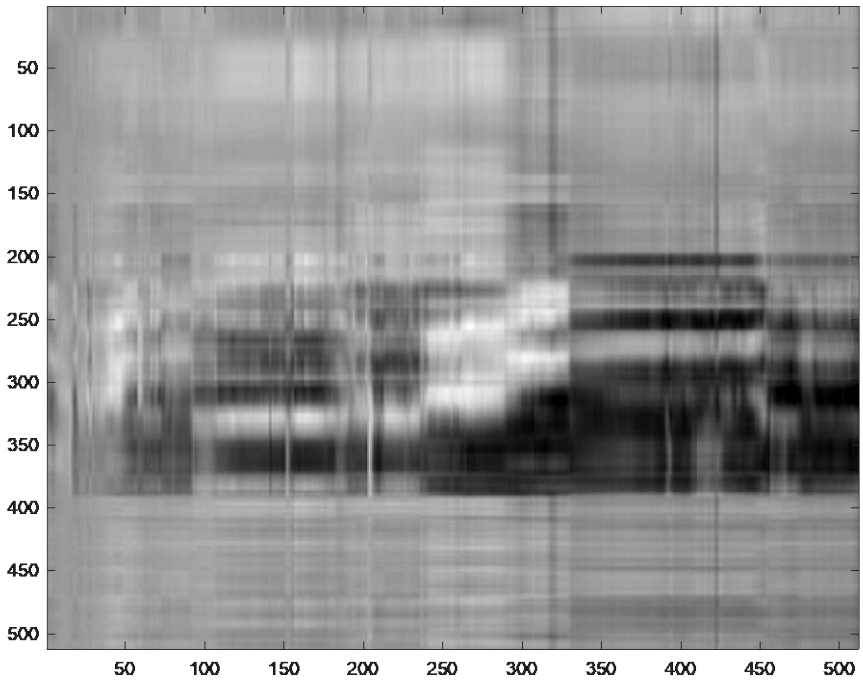
File: proj_and_rec.m

```
% Matlab function to project and recover image data
function recoverImageMatByN = proj_and_rec(x,pcvecs,n)
    % Projection of normalized immat on PC
    projimgN=x*pcvecs(:,1:n);
```

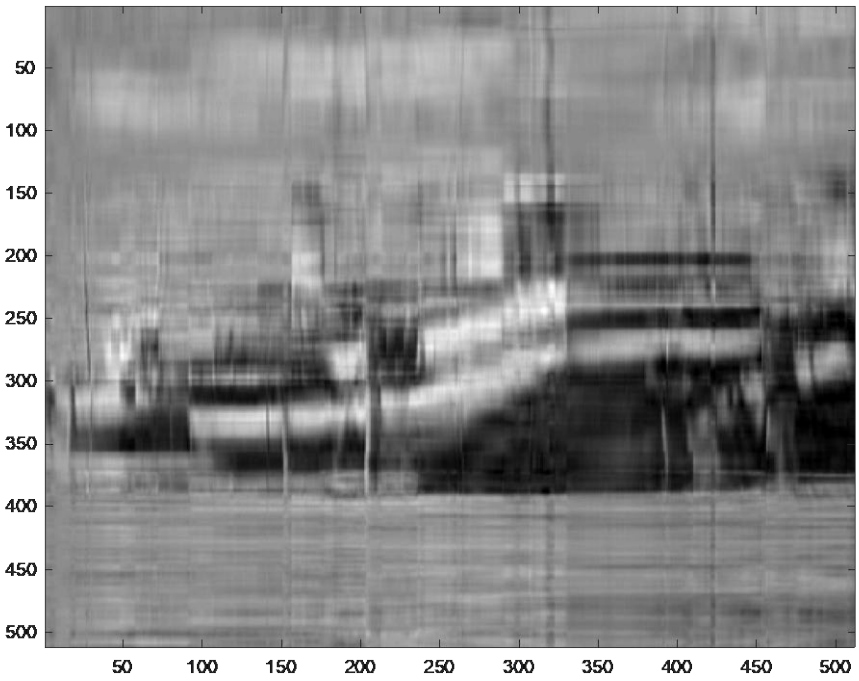
```
    % Select first n pc
    firstNpc=pcvecs(:,1:n);
    % Recover image matrix using first n PC
    recoverImageMatByN=projimgN*firstNpc';
end
```

Output Images/Fig obtained

1. Using first 4 PC



2. Using first 10 PC



3. Using first 15 PC

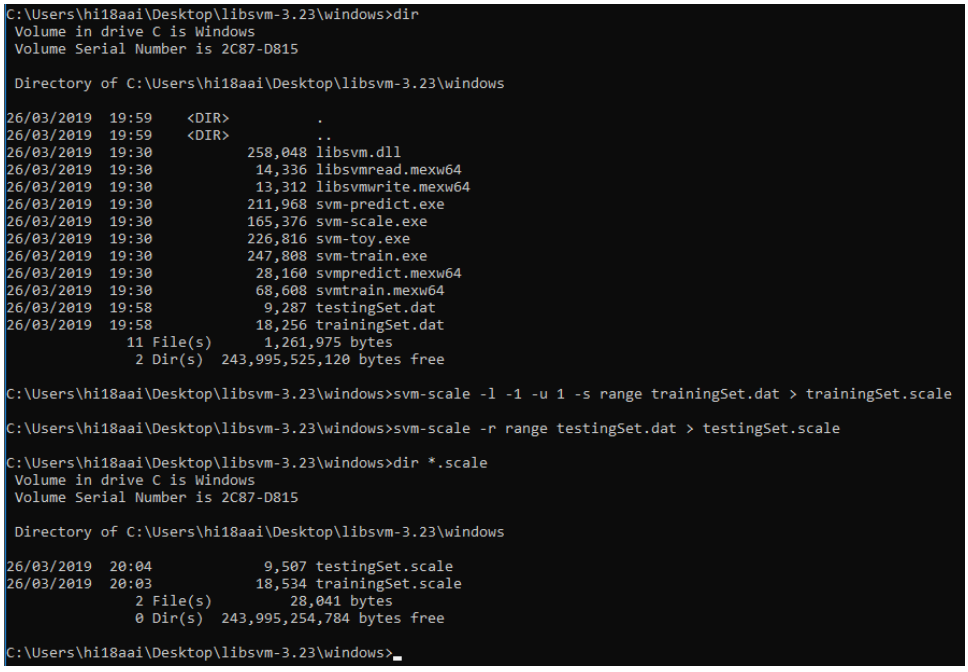
Critical Analysis and Summary

Answer 4

On performing PCA on the boat image's matrix data, we obtain eigen vector (a 512x512 matrix). We transform the original matrix data to PC coordinate system by taking all 512 rows and first 4,10,15 columns (PC components) respectively. As a result, we obtain 512x4,512x10 and 512x15 matrices which can be represented in PC coordinate system. For recovering we multiply this projected data with transpose of the slice of eigen vector which uses first 4, 10 and 15 PC components. Considering first 4 PC components, by Matrix-multiplication of projected matrix 512x4 with 4x512 (transpose of eigenvectors first 4 columns), we obtain the image which looks similar to original image but with less details. Higher the number of PC components we use, better the quality image recovered. To obtain exact original image, we must use all the PC components. Anything between 1-254 will give us an image with less details.
To obtain the image in original feature space, we should have matrix data in the range of 0-255 (negative integers do not represent a valid pixels). For the same elementwise multiplication of standard deviation value should be done on recovered data and the mean subtracted during normalizing the data before performing PCA should also be added.

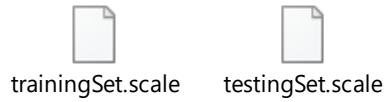
Task 3 – Classification using SVM

Normalization



Attached scaled/normalized files
Command svm-scale is used for scaling training and testing data set -
svm-scale -l -1 -u 1 -s range trainingSet.dat > trainingSet.scale
Command for testing set -
Method A) svm-scale -r range testingSet.dat > testingSet.scale
or

Method B) svm-scale -l -1 -u 1 -s testingSet.dat > testingSet.scale



Cross Validation

For 5-fold cross-validation we will use command svm-train with option –v 5
svm-train -c 100 -g 5 -v 5 trainingSet.scale
svm-train -c 10 -g 3 -v 5 trainingSet.scale
svm-train -c 5 -g 0.3 -v 5 trainingSet.scale
Below are the Cross-Validation Accuracy results of given combinations of cost and gamma.

Cost (C)	Gamma (γ)	Cross Validation Accuracy
100	5	65.2542%
10	3	86.4407%
5	0.3	95.7627%

From above results it can be concluded that C = 5 and Gamma = 0.3 is the optimal combination.

Classification

First, we will perform training based on the best Cost and Gamma combination, i.e., C = 5 and Gamma = 0.3
svm-train -c 5 -g 0.3 trainingSet.scale trainingSet.model

```
C:\Users\hi18aai\Desktop\libsvm-3.23\windows>svm-train -c 5 -g 0.3 trainingSet.scale trainingSet.model
*
optimization finished, #iter = 60
nu = 0.065707
obj = -14.635491, rho = 0.879221
nSV = 19, nBSV = 1
*
optimization finished, #iter = 42
nu = 0.020018
obj = -3.553104, rho = 0.112642
nSV = 17, nBSV = 0
*.*
optimization finished, #iter = 81
nu = 0.061076
obj = -12.063906, rho = -0.229927
nSV = 23, nBSV = 0
Total nSV = 44
```

Output Prediction on testingSet.scale using Method A
svm-predict testingSet.scale trainingSet.model predicted.output

```
C:\Users\hi18aai\Desktop\libsvm-3.23\windows>svm-predict testingSet.scale trainingSet.model predicted.output
Accuracy = 100% (60/60) (classification)
```

Output Prediction on testingSet.scale using Method B
svm-predict testingSet.scale trainingSet.model predicted.output

```
C:\Users\hi18aai\Desktop\libsvm-3.23\windows>svm-predict testingSet.scale trainingSet.model predicted.output
Accuracy = 96.6667% (58/60) (classification)
```

Task 4 – Classification Analysis

Misclassified instances can be identified by comparing the predicted.output from task 3 and the given labels - tstlabels loaded using wine.mat
For PCA, pcvecs and pcvals are taken from wine_pca.mat saved in Task1.
2 of the instances from test set are found to be predicted incorrectly as class 3, but they are actually class 2. Below program is used to plot the misclassified instances.

Source File: analyze_misclassified.m

```
% Load actual data/given labels
load wine.mat
% Load predicted labels from task 3
load predicted.output
% Get tstdata projected on Principal components
mean_trndata=mean(trndata);
mean_tstdata=mean(tstdata);
std_trndata=std(trndata);
std_tstdata=std(tstdata);
norm_trndata=(trndata-mean_trndata)./std_trndata;
norm_tstdata=(tstdata-mean_tstdata)./std_tstdata;
% Perform PCA using training data set
[pcvals,pcvecs]=pca(norm_trndata);
% Project test data set
projtstdata = norm_tstdata*pcvecs(:,1:2);
figure(1)
hold on
h1=plot(projtstdata(tstlabels==1,1),projtstdata(tstlabels==1,2),'r.','Marke
rSize',10);
h2=plot(projtstdata(tstlabels==2,1),projtstdata(tstlabels==2,2),'b.','Marke
rSize',10);
h3=plot(projtstdata(tstlabels==3,1),projtstdata(tstlabels==3,2),'g.','Marke
rSize',10);
h4=plot(projtstdata(tstlabels~=predicted,1),projtstdata(tstlabels~=predicte
d,2),'kx','MarkerSize',10);
legend('Class 1','Class 2','Class 3','Misclassified');
xlabel('PC1');
ylabel('PC2');
set(gca,'Box','on')
```

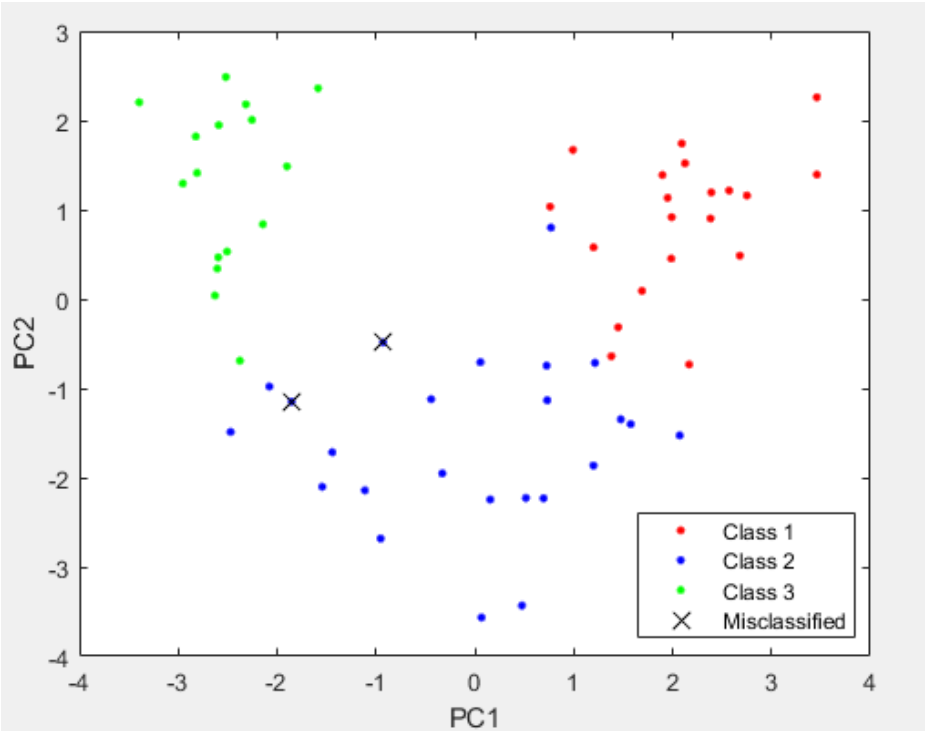


Figure – 3 Misclassified Data Plot from Test set

Critical Analysis

Instance are misclassified because they either didn't fall in the feature space of the different category identified by SVM. Misclassified data set could be a result of underfitting or overfitting.

Task 5 – Analysis using R

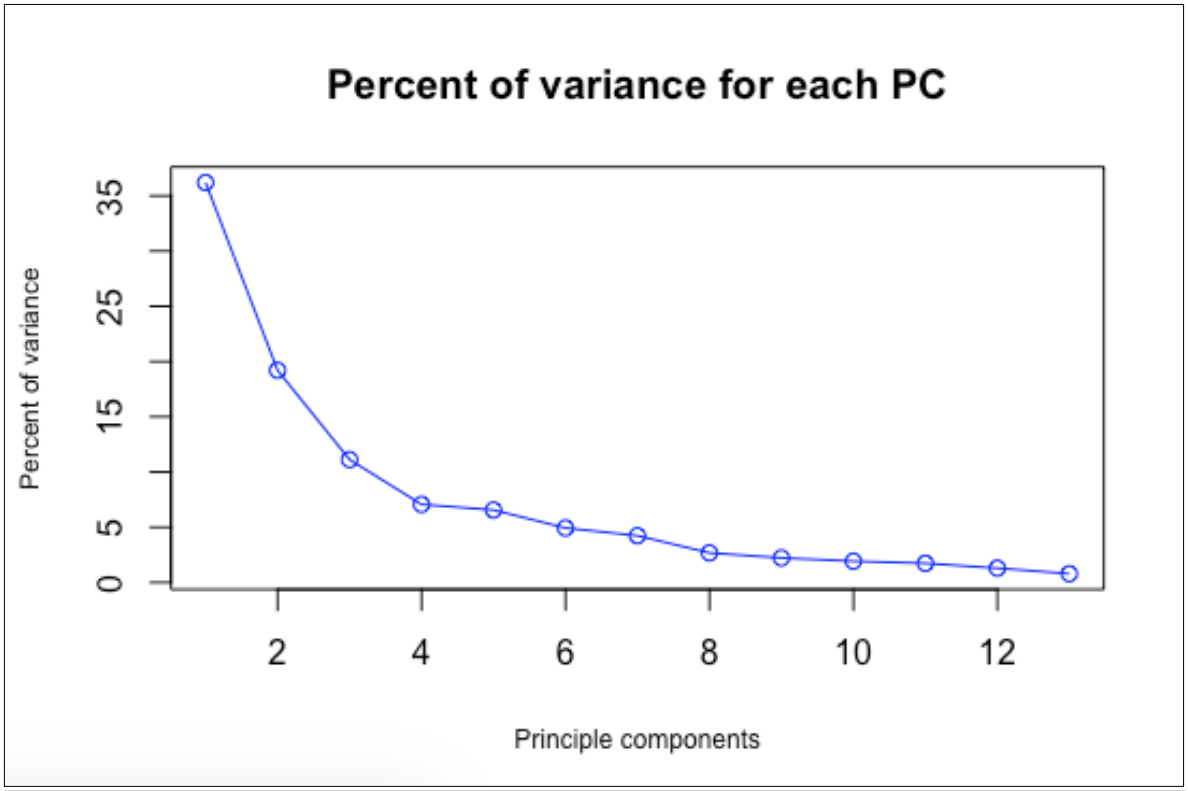
Source Code: File : Task5.R

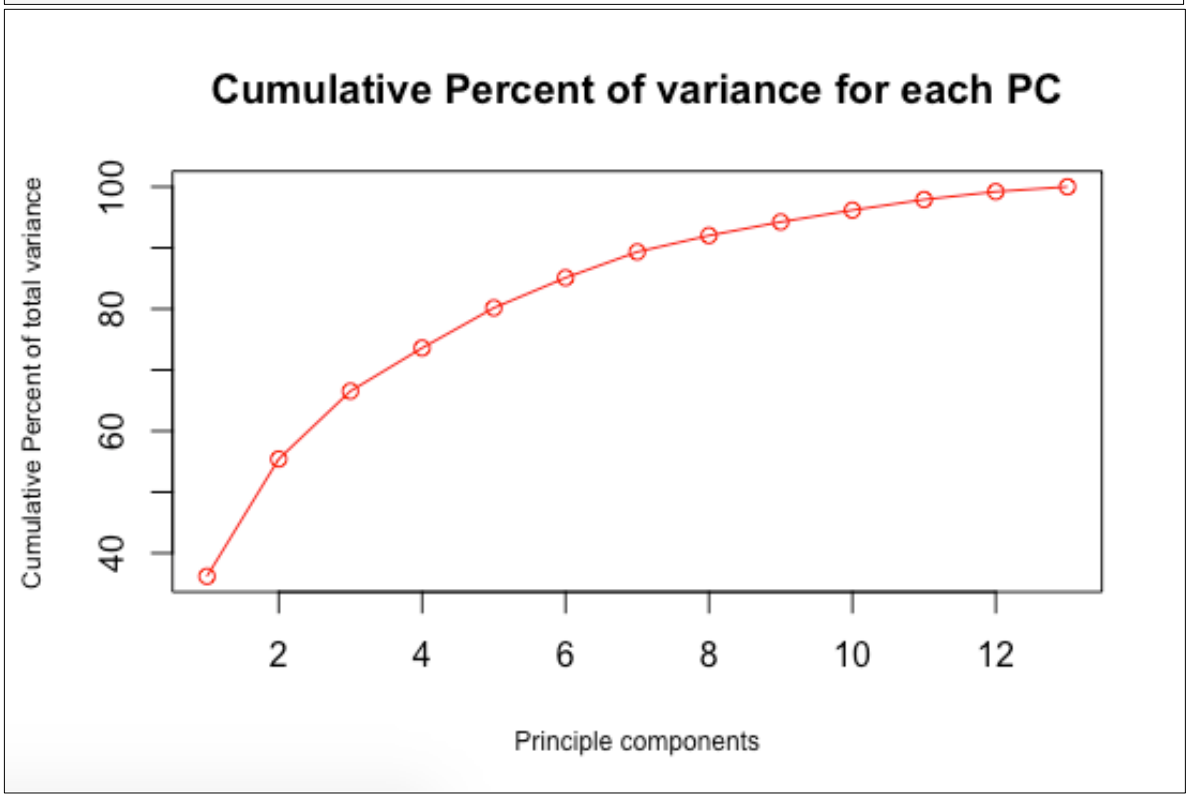
```
# read wine data from file. Edit the wine.dat to add column names in first line
wine.table=read.delim("wine.data.txt",sep=",")
# Take columns 2 to 14 for analysis. Ignoring first column, class labels here.
wine_data=wine.table[2:14]
#attach(wine_data)
# Scale wine data
wine_data.s=scale(wine_data,center=TRUE,scale=TRUE)
# Find correlation values
wine_data.cor=cor(wine_data)
wine_data.cor
# Find eigen vectors and eigen values
```

```
wine_data.eig=eigen(wine_data.cor)
# Extract eigen values
wine_data.eigval=cbind(wine_data.eig$values)
wine_data.eigval
# find sum of eigenv values
sum(wine_data.eigval)
# Proportion of wine data for each PC
pr_wine_data.eigenval=wine_data.eigval/sum(wine_data.eigval)
pr_wine_data.eigenval
pr_wine_data.eigenvectors=cbind(wine_data.eig$vectors)
pr_wine_data.eigenvectors
# Cumulative sum of of eigen vals
cumsum(pr_wine_data.eigenval)
# Plot of how much percentage each principle component covers.
plot(pr_wine_data.eigenval*100,
    ylab="Percent of variance",
    xlab="Principle components",
    cex.lab=0.75,col="blue",type="o",
    main="Percent of variance for each PC")
# Plot of cumulative sum of principle components.
# First 4 components explain upto 73% of the data
# First 5 components explain up 80% of the data
plot(cumsum(pr_wine_data.eigenval)*100,
    ylab="Cumulative Percent of total variance",
    xlab="Principle components",
    cex.lab=0.75,col="red",type="o",
    main="Cumulative Percent of variance for each PC")

# Perform hierarichal clustering analysis.
# First we will project the given data in PC coordinate system
# by doing matrix multiplication of scaled wine data with eigen vector
wine_data.new=wine_data.s%*%pr_wine_data.eigenvectors
# Add column names
colnames(wine_data.new)=c("PC1","PC2","PC3","PC4","PC5","PC6","PC7","PC8","PC9","PC1
0","PC11","PC12","PC13")
# Clustering reduced number of dimensions, using first 2 PCs
wine_data.RD=wine_data.new[,c(1,2)]
# Clustering reduced number of dimensions, using first 4 PCs
#wine_data.RD=wine_data.new[,c(1,2,3,4)]
wineclust=hclust(dist(wine_data.RD),method="ward.D2")
# store the labels, for later use
X = t(wine.table[,1])
# Create denodgram
plot(wineclust,labels=X,cex=0.40,hang=-2)
```

- Scree Plot

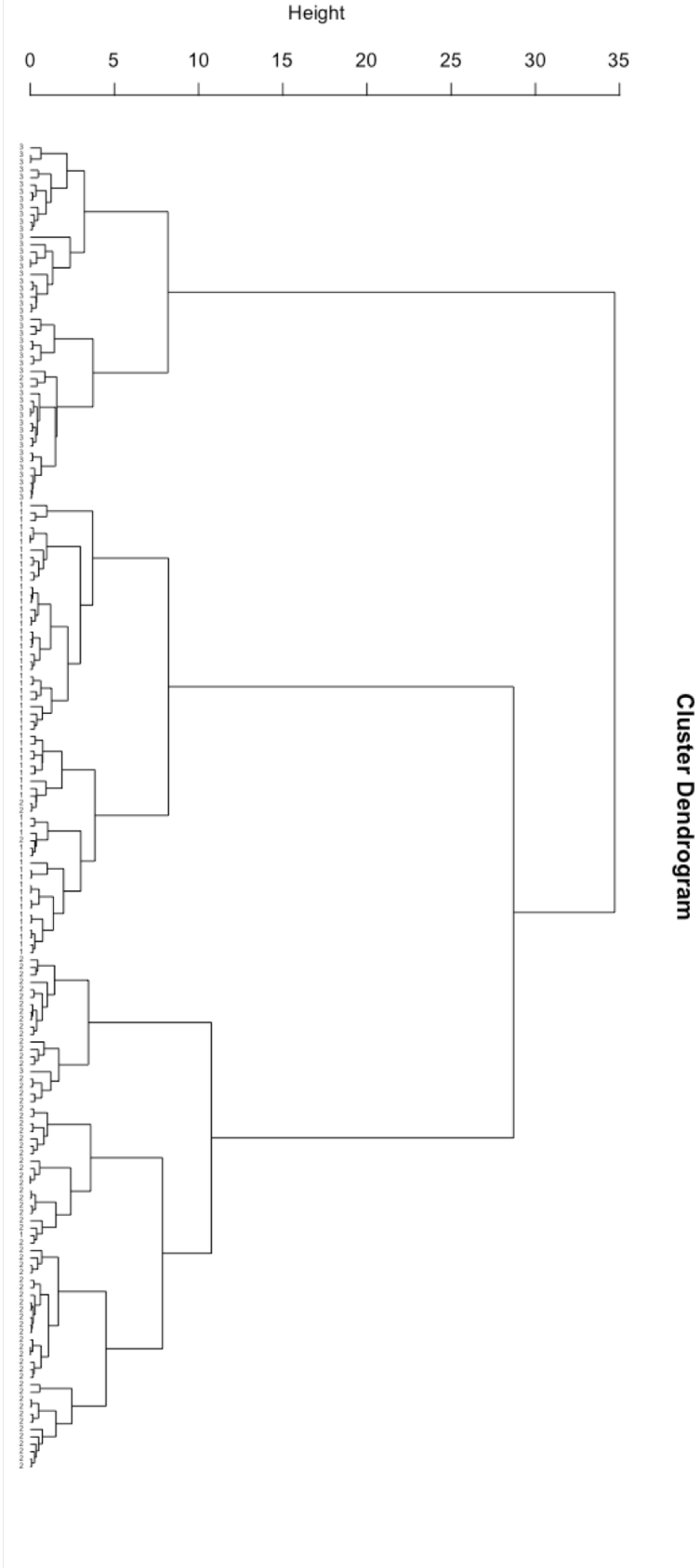




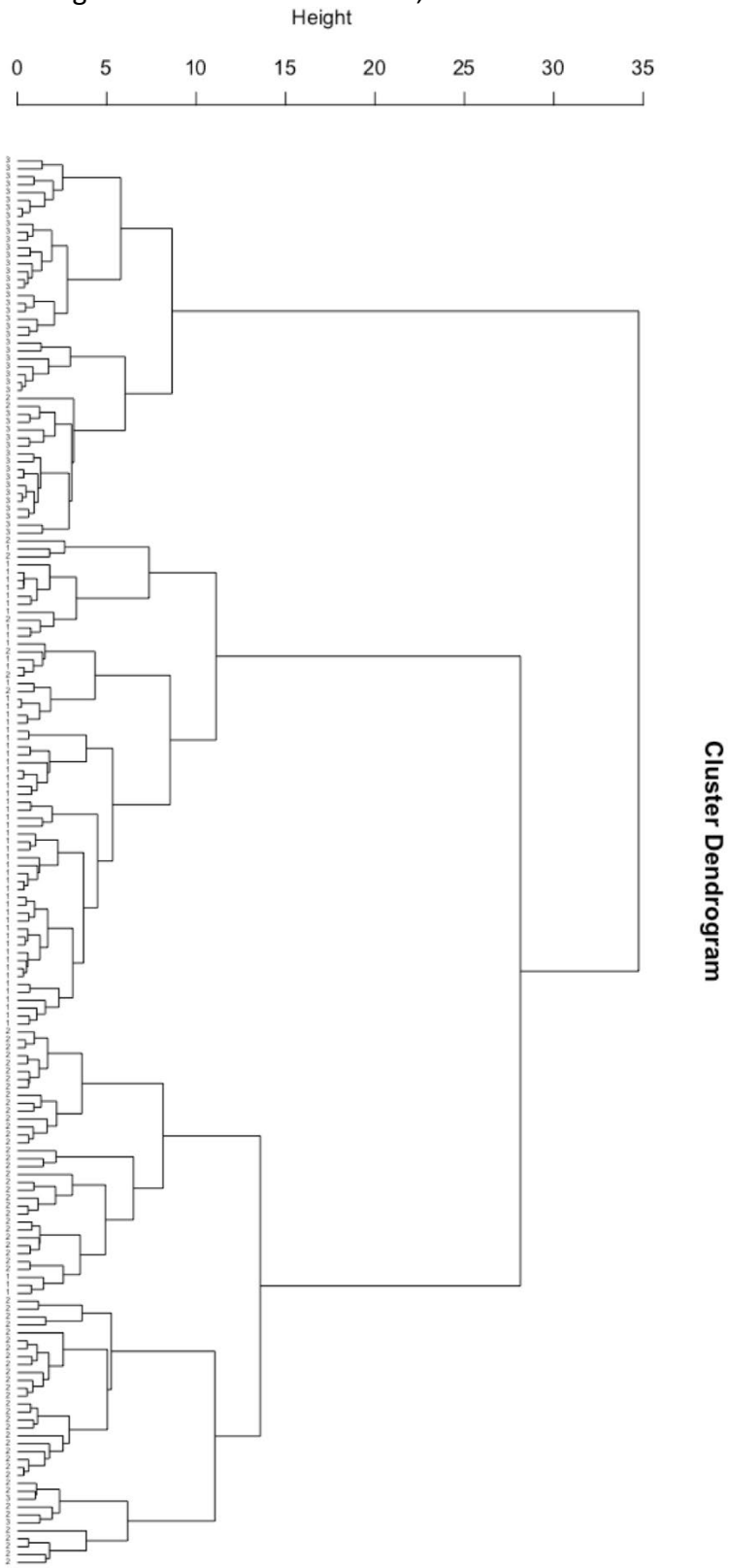
- From the above cumulative plot, it can be concluded that first 4 PC components almost 75% of the variance.

Together they cover 73.5% of total variance.

- From the scree plot, the first 2 Principal Components explain only 55.4 % of the variance.
- Dendrogram-1 based on first 2 PCs



- Dendrogram-2 based on first 4 PCs, as it covers more % of variance



- Critical analysis and comparison with MATLAB-PCA and K-means
  Below is the comparison of eigen vectors obtained after PCA in MATLAB and R.
  Eigen vector obtained in R

1	-0.144329395	-0.483651548	-0.20738262	-0.01785630	-0.26566365	0.21353865	0.05639636	0.39613926	0.50861912	0.21160473	-0.22591696	-0.26628645	-0.01496997
2	0.245187580	-0.224930935	0.08901289	0.53689028	0.03521363	0.53681385	-0.42052391	0.06582674	-0.07528304	-0.30907994	0.07648554	0.12169604	-0.02596375
3	0.002051061	-0.316068814	0.62622390	-0.21417556	-0.14302547	0.15447466	0.14917061	-0.17026002	-0.30769445	-0.02712539	-0.49869142	-0.04962237	0.14121803
4	0.239320405	0.010590502	0.61208035	0.06085941	0.06610294	-0.10082451	0.28696914	0.42797018	0.20044931	0.05279942	0.47931378	-0.05574287	-0.09168285
5	-0.141992042	-0.299634003	0.13075693	-0.35179658	0.72704851	0.03814394	-0.32288330	-0.15636143	0.27140257	0.06787022	0.07128891	0.06222011	-0.05677422
6	-0.394660845	-0.065039512	0.14617896	0.19806835	-0.14931841	-0.08412230	0.02792498	-0.40593409	0.28603452	-0.32013135	0.30434119	-0.30388245	0.46390791
7	-0.422934297	0.003359812	0.15068190	0.15229479	-0.10902584	-0.01892002	0.06068521	-0.18724536	0.04957849	-0.16315051	-0.02569409	-0.04289883	-0.83225706
8	0.298533103	-0.028779488	0.17036816	-0.20330102	-0.50070298	-0.25859401	-0.59544729	-0.23328465	0.19550132	0.21553507	0.11689586	0.04235219	-0.11403985
9	-0.313429488	-0.039301722	0.14945431	0.39905653	0.13685982	-0.53379539	-0.37213935	0.36822675	-0.20914487	0.13418390	-0.23736257	-0.09555303	0.11691707
10	0.088616705	-0.529995672	-0.13730621	0.06592568	-0.07643678	-0.41864414	0.22771214	-0.03379692	0.05621752	-0.29077518	0.03183880	0.60422163	0.01199280
11	-0.296714564	0.279235148	0.08522192	-0.42777141	-0.17361452	0.10598274	-0.23207564	0.43662362	0.08582839	-0.52239889	-0.04821201	0.25921400	0.08988884
12	-0.376167411	0.164496193	0.16600459	0.18412074	-0.10116099	0.26585107	0.04476370	-0.07810789	0.13722690	0.52370587	0.04642330	0.60095872	0.15671813
13	-0.286752227	-0.364902832	-0.12674592	-0.23207086	-0.15786880	0.11972557	-0.07680450	0.12002267	-0.57578611	0.16211600	0.53926983	-0.07940162	-0.01444734

Eigen vector obtained in MATLAB

pcvecs

13x13 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.1443	0.4837	0.2074	-0.0179	0.2657	-0.2135	-0.0564	0.3961	-0.5086	-0.2116	-0.2259	0.2663	0.0150
2	-0.2452	0.2249	-0.0890	0.5369	-0.0352	-0.5368	0.4205	0.0658	0.0753	0.3091	0.0765	-0.1217	0.0260
3	-0.0021	0.3161	-0.6262	-0.2142	0.1430	-0.1545	-0.1492	-0.1703	0.3077	0.0271	-0.4987	0.0496	-0.1412
4	-0.2393	-0.0106	-0.6121	0.0609	-0.0661	0.1008	-0.2870	0.4280	-0.2004	-0.0528	0.4793	0.0557	0.0917
5	0.1420	0.2996	-0.1308	-0.3518	-0.7270	-0.0381	0.3229	-0.1564	-0.2714	-0.0679	0.0713	-0.0622	0.0568
6	0.3947	0.0650	-0.1462	0.1981	0.1493	0.0841	-0.0279	-0.4059	-0.2860	0.3201	0.3043	0.3039	-0.4639
7	0.4229	-0.0034	-0.1507	0.1523	0.1090	0.0189	-0.0607	-0.1872	-0.0496	0.1632	-0.0257	0.0429	0.8323
8	-0.2985	0.0288	-0.1704	-0.2033	0.5007	0.2586	0.5954	-0.2333	-0.1955	-0.2155	0.1169	-0.0424	0.1140
9	0.3134	0.0393	-0.1495	0.3991	-0.1369	0.5338	0.3721	0.3682	0.2091	-0.1342	-0.2374	0.0956	-0.1169
10	-0.0886	0.5300	0.1373	0.0659	0.0764	0.4186	-0.2277	-0.0338	-0.0562	0.2908	0.0318	-0.6042	-0.0120
11	0.2967	-0.2792	-0.0852	-0.4278	0.1736	-0.1060	0.2321	0.4366	-0.0858	0.5224	-0.0482	-0.2592	-0.0899
12	0.3762	-0.1645	-0.1660	0.1841	0.1012	-0.2659	-0.0448	-0.0781	-0.1372	-0.5237	0.0464	-0.6010	-0.1567
13	0.2868	0.3649	0.1267	-0.2321	0.1579	-0.1197	0.0768	0.1200	0.5758	-0.1621	0.5393	0.0794	0.0144

It can be seen that R Eigen vector is almost same as that of MATLAB, when rounded to 4 digits after decimal. This means there is a lot of similarity in the PCA done on MATLAB and R.

Task 1 - part 1 principal component analysis show that the 3 cultivators (class) of wine are somewhat distinct/separated in the feature space, which indicate they might have different combination of chemicals/features given. Though it is based on only first 2 PCs. The same can be inferred using the dendrogram-1 created using first 2 PCs.

In task 1 - part 2, plots are created based on clusters resulted from k-means with K=3 and K=5, shown in Figure 11 and Figure 12 respectively. It is also done using first 2 principal components.
Looking at the dendrogram-1 created using 2 PCs from task 5, it can be observed that most of the wine data samples of classes 1, 2 and 3 are closed related to their own respective samples. However on a macro level, the group of samples of 1 as a whole are related to group of samples of 2 as a whole, and 1 and 2 combined are related to group of samples of 3. Although there are some instances where samples of 1 and 2 are closely related to each other. This can also be seen in the CPA plot. Based on dendrograms created in task5, it is more clearly evident that wine data can be classified as 3 clusters rather than 5.

Advantages of Hierarchal Cluster Analysis over K-means is that when we perform Hierarchal clustering guess work, trial and error method is not need to get the correct classification. For example, from dendrogram it can clearly observed on a macro level that - how many clusters of related samples we have on a high level,

having a large distance between them. In the given wine data scenario, we have 3 major clusters as seen from dendrogram. We may use this info (as an initial info, to begin with) to consider an appropriate number of code-vectors/centroids/k-value before performing the k-means.
Hierarchal cluster analysis using dendrogram can also be useful in finding out which two samples are closely related to each other, which may not be identifiable by looking at data points projected on a graph on PC1-PC2 axes.