STATISTICAL METHODS IN AI
ASSIGNMENT5:
DIMENSIONALITY REDUCTION AND CLUSTERING

Name – Megha Agarwal
Roll No – 201506511
Course – M.Tech CSIS(PG1)
Dated : 30 March, 2016

AIM:
The aim of this assignment is to experiment with dimensionality reduction and clustering techniques we learned in the class on real world problems.

DATA SETS DESCRIPTION :

Data Set : IRIS DATA SET

Number of Instances: 150 (50 in each of three classes)

- ➔ Title: Iris Plant DataSet
- ➔ Number of Attributes: 4 numeric, predictive attribues and the class.
- ➔ Attribute information:
    1. sepal length in cm
    2. sepal width in cm
    3. petal length in cm
    4. petal width in cm
    5. class:
        -- Iris Setosa
        -- Iris Versicolour
        -- Iris Virginica

- ➔ Missing Attribute Values: None
- ➔ Class Distribution : 33.33% for each of 3 classes.

CODE :

A) Dimensionality Reduction

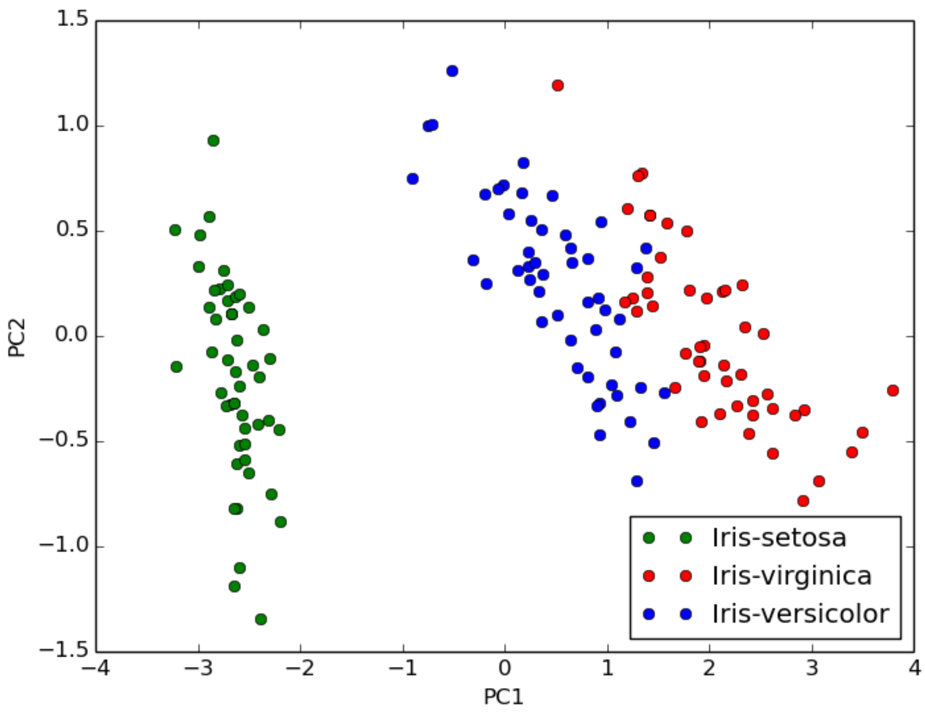
PCA on Iris Data Set :

```
import numpy as np
import matplotlib.pyplot as plt

def read_file_load_dataset(filename, index_of_class):
	f = open(filename, 'rb')
	dataset = f.readlines()
	length = len(dataset)
	sample=[]
	for i in range(0, length):
		test = dataset[i].split(',')
		for j in range(0, len(test)-1):
			test[j] = float(test[j])
		sample.append(test[:-1])

	return dataset, sample

def calculating_pca(sample):
	scatter=np.dot(sample,sample.T)
	eigenvalue, eigenvector=np.linalg.eig(scatter)

	eigenpairs=[]
	for i in range(len(eigenvalue)):
		eigenpairs.append((np.abs(eigenvalue[i]), eigenvector[:,i]))

	pca1 = eigenpairs[0][1].reshape(4,1)
	pca2 = eigenpairs[1][1].reshape(4,1)
	pca2 = eigenpairs[1][1].reshape(4,1)
	pca3 = eigenpairs[2][1].reshape(4,1)

	pca_temp1 = np.hstack((pca1, pca2))
	pca_temp2 = np.hstack((pca2, pca3))
	pca_temp3 = np.hstack((pca1, pca3))

	pca12=np.dot(pca_temp1.T,sample)
	pca23=np.dot(pca_temp2.T,sample)
	pca31=np.dot(pca_temp3.T,sample)


	return pca12, pca23, pca31

def save_plot(pca_plot,label_1,label_2,name):

	plt.ylabel(label_1)
	plt.xlabel(label_2)
	class1=plt.plot(pca_plot[0,:50],pca_plot[1,:50],'go',label="Iris-setosa")
	class3=plt.plot(pca_plot[0,100:151],pca_plot[1,100:151],'ro',label="Iris-virginica")
	class2=plt.plot(pca_plot[0,50:100],pca_plot[1,50:100],'bo',label="Iris-versicolor")

	location='lower right'
	plt.legend(loc=location)
	plt.savefig(name)
	plt.close()


def plot(pca12, pca23, pca31):
	save_plot(pca12,'PC2','PC1','pca12.png')
	save_plot(pca23,'PC3','PC2','pca23.png')
	save_plot(pca31,'PC3','PC1','pca13.png')


if __name__ == '__main__':

	filename = "iris.data"
	index_of_class=4
	dataset, sample = read_file_load_dataset(filename, index_of_class)
	sample = np.array(sample)
	sample = np.transpose(sample)

	feature1,feature2,feature3, feature4 = sample[0], sample[1], sample[2], 
sample[3]
	mean1, mean2, mean3, mean4 = np.mean(feature1), np.mean(feature2), 
np.mean(feature3), np.mean(feature4)

	sample[0] = sample[0] - mean1
	sample[1] = sample[1] - mean2
	sample[2] = sample[2] - mean3
	sample[3] = sample[3] - mean4

	pca12, pca23, pca31 = calculating_pca(sample)
	plot(pca12, pca23, pca31)
```

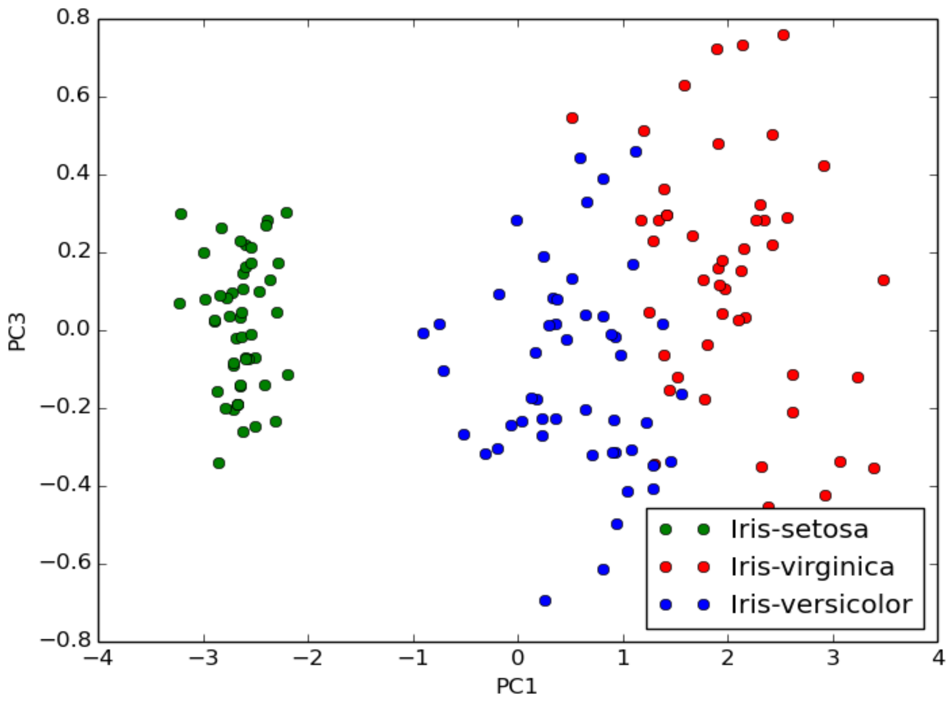
SAMPLE OUTPUT:

Projection of the original data in PCA space

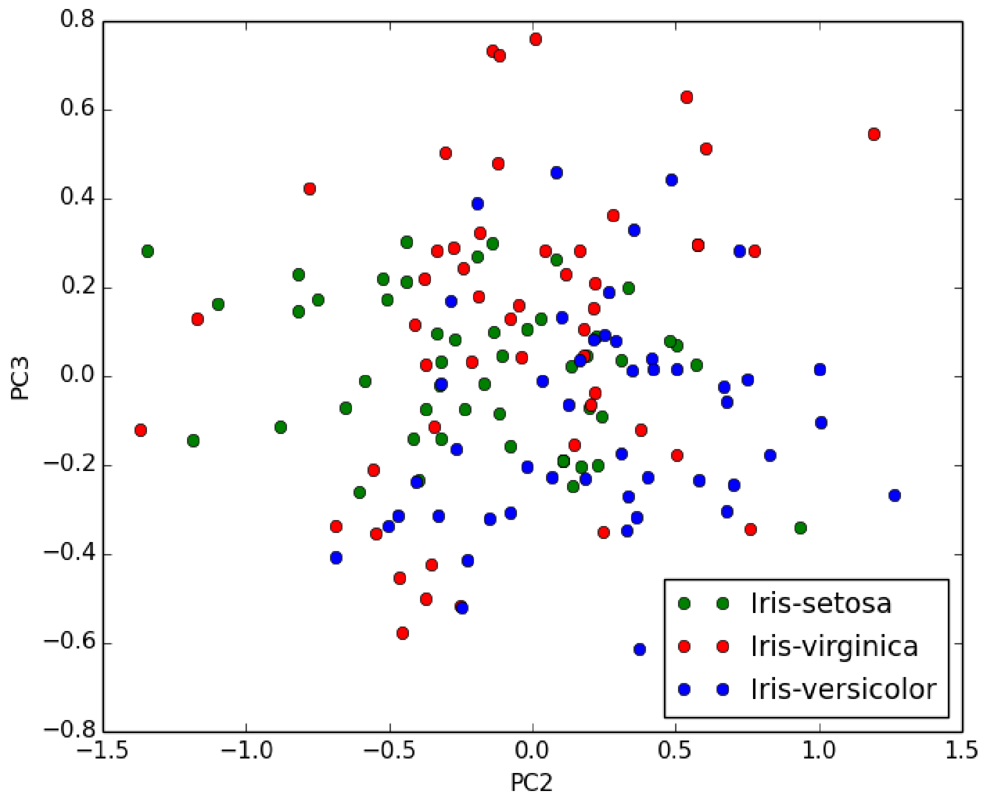
PC1 versus PC2



PC1 versus PC3



PC2 versus PC3



PCA on Arcene Data Set (Data Set with large Dimensions) :

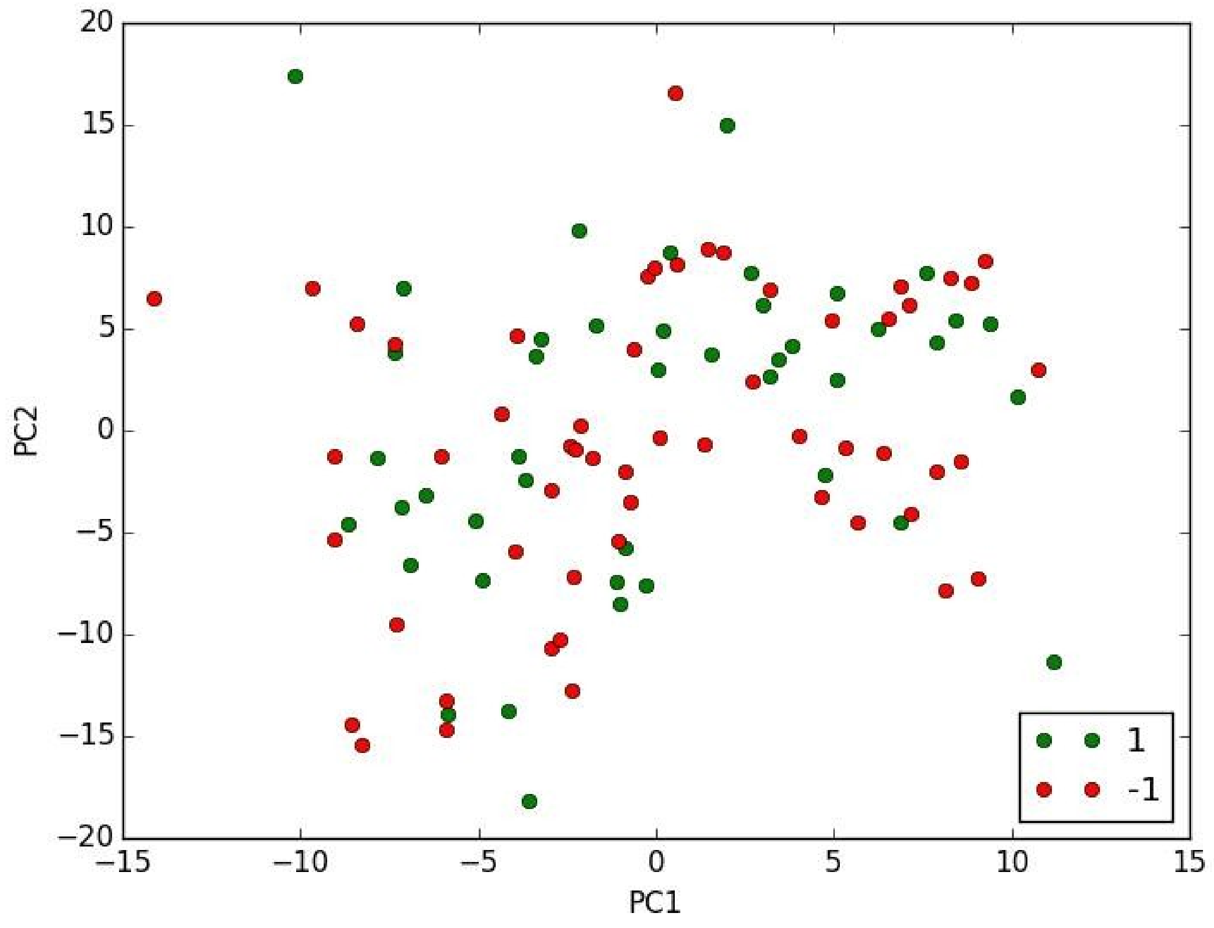
```
import numpy as np
import matplotlib.pyplot as plt
import csv

def read_file_load_dataset(filename):
        f = open(filename, 'rb')
        dataset = f.readlines()
        length = len(dataset)

        sample=[]
        for i in range(0, length):
                test = dataset[i].split(' ')
                for j in range(0, len(test)-1):
                        test[j] = float(test[j])
                sample.append(test[:-1])

        return dataset, sample


def plot(x1, y1, x2, y2, xlabel, ylabel, name):
        plt.ylabel(label_1)
```

```
    plt.xlabel(label_2)
    class2=plt.plot(x2,y2,'bo',label="-1")
    class1=plt.plot(x1,y1,'ro',label="1")
    location='lower right'
    plt.legend(loc=location)
    plt.savefig(name)
    plt.close()


def plot2(x, eigenvalue, xlabel, ylabel, name):
    plt.ylabel(label_1)
    plt.xlabel(label_2)
    class1=plt.plot(x,eigenvalue)
    location='lower right'
    plt.legend(loc=location)
    plt.savefig(name)
    plt.close()


def calculating_eigenpairs(sample):
    scatter=np.dot(sample,sample.T)
    print scatter.shape
    eigenvalue, eigenvector=np.linalg.eig(scatter)
    eigenpairs=[]

    for i in range(len(eigenvalue)):
        eigenpairs.append((np.abs(eigenvalue[i]), eigenvector[:,i]))

    index=np.argsort(eigenvalue)
    vector1, vector2 =eigenvector[index[-1]], eigenvector[index[-2]]
    vector1 = vector1.reshape(10000,1)
    vector2 = vector2.reshape(10000,1)

    pc12 = np.hstack((vector1, vector2))
    pca=np.dot(pc12.T,sample)

    sorted(eigenvalue,reverse=True)
    return eigenvalue, pca


if __name__ == '__main__':

    filename = "arcene_train.data"
    dataset, sample = read_file_load_dataset(filename)
    sample = np.transpose(sample)

    with open('arcene_train.labels','rb') as f:
        reader=csv.reader(f)
        labels=list(reader)
    x1,y1, x2, y2 = [],[],[],[]
```

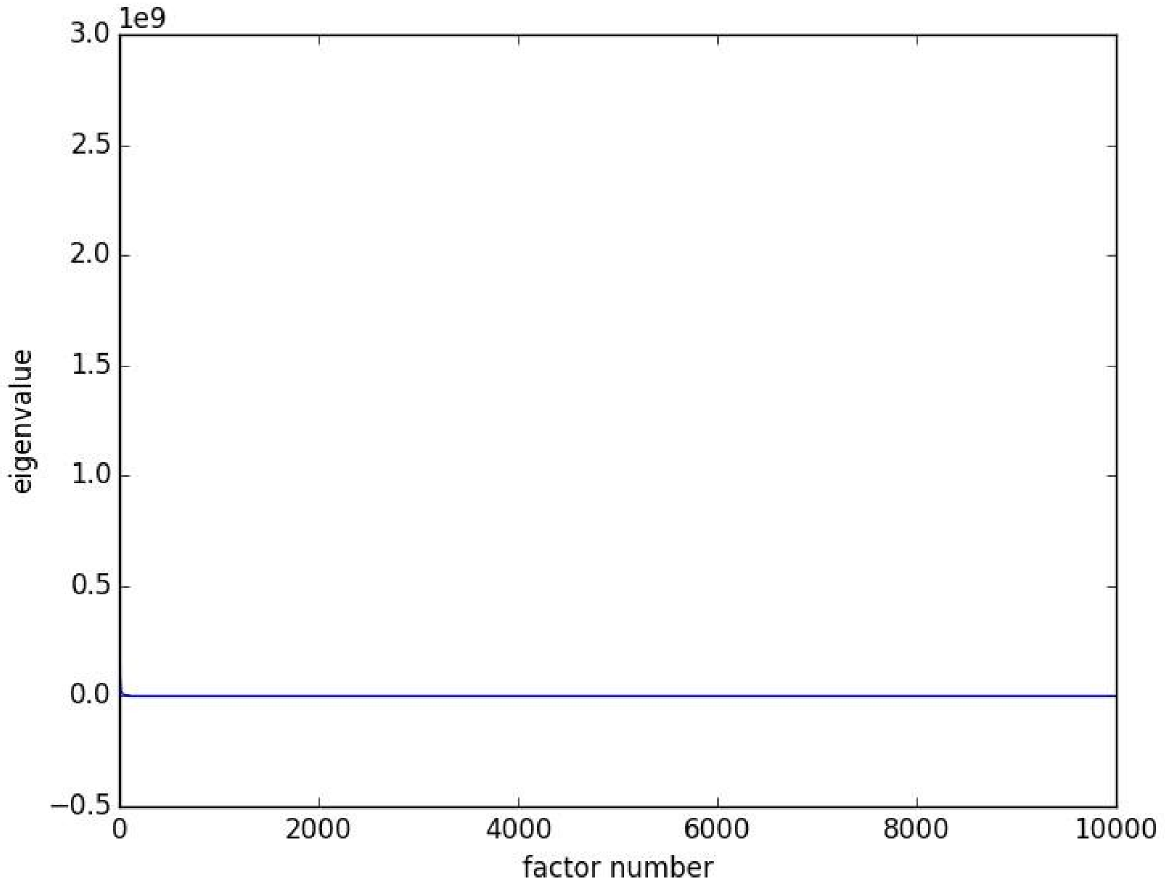
```
ar = 10000*[0.0]
avg = np.array(ar)
for i in range(0, 10000):
        avg[i]=np.mean(sample[i])

for i in range(0, 10000):
        for j in range(len(sample[i])):
                sample[i][j]=sample[i][j]-avg[j]

eigenvalue, pca = calculating_eigenpairs(sample)

for i in range(0, 100):
        if labels[i]==['1']:
                x1.append(pca[0][i])
                y1.append(pca[1][i])

for i in range(0, 100):
        if labels[i]==['-1']:
                x2.append(pca[0][i])
                y2.append(pca[1][i])

plot(x1, y1, x2, y2, 'PC1', 'PC2', "pca_large_data12.png")
x=[]
for i in range(1,10001):
        x.append(i)


plot2(x, eigenvalue, 'factor_number', 'eigenvalue', 'screeplot.png')
```

SAMPLE OUTPUT:
Projection of the original data in PCA space

PC1 versus PC2



Screeplot



How many components to choose for explaining 85%, 90%, 95% and 99% of the variance of the data.

The large dataset is producing (10000) eigenvalues.

Procedure to get the defined accuracy :

- Perform the summation of all eigenvalues generated.
- Divide the eigenvalues by the sum generated.
- The difference from 1.00 gives the loss percentage by that particular eigenvalue. i.e. if the ratio so obtained is .82 then this implies that the eigenvalue is able to campture 82% variance of the data and 8% is the loss.
- So, as to make the variance 85%, we need to choose a component which capture 3% of the data, making to the sum of 85%.
- To make it further to 90%, we will add more eigenvalues, and finally more to make it 99%.
- Thus, it suggestes that we can be selective in choosing the eigenvalues to reduce dimensions so as to produces the dataset of the required variance i.e. 85%, 90%, 95%, 99%.

LDA on Iris Data Set :

```
import numpy as np
import matplotlib.pyplot as plt

def read_file_load_dataset(filename, index_of_class, m):
        f = open(filename, 'rb')
        dataset = f.readlines()
        length = len(dataset)
        sample=[]
        for i in range(0, length):
                test = dataset[i].split(',')
                for j in range(0, len(test)-1):
                        test[j] = float(test[j])
                sample.append(test[:-1])

        s1 = sample[:50]
        s2 = sample[50:100]
        s3 = sample[100:]

        sample1=np.asarray([0])
        for i in range(len(s1)):
                for j in range(0, len(s1[i])):
                        m[0][j]+=float(s1[i][j])
                s1 = np.array(s1)
                sample1 = np.concatenate([sample1,s1[i]])
        sample1=np.delete(sample1,0)

        sample2=np.asarray([0])
        for i in range(len(s2)):
                for j in range(0, len(s2[i])):
                        m[1][j]+=float(s2[i][j])
                s2 = np.array(s2)
                sample2 = np.concatenate([sample2,s2[i]])
        sample2=np.delete(sample2,0)


        sample3=np.asarray([0])
        for i in range(len(s3)):
                for j in range(0, len(s3[i])):
                        m[2][j]+=float(s3[i][j])
                s3 = np.array(s3)
                sample3 = np.concatenate([sample3,s3[i]])
        sample3=np.delete(sample3,0)
        m=m/float(50)
        return sample1, sample2, sample3, m
```

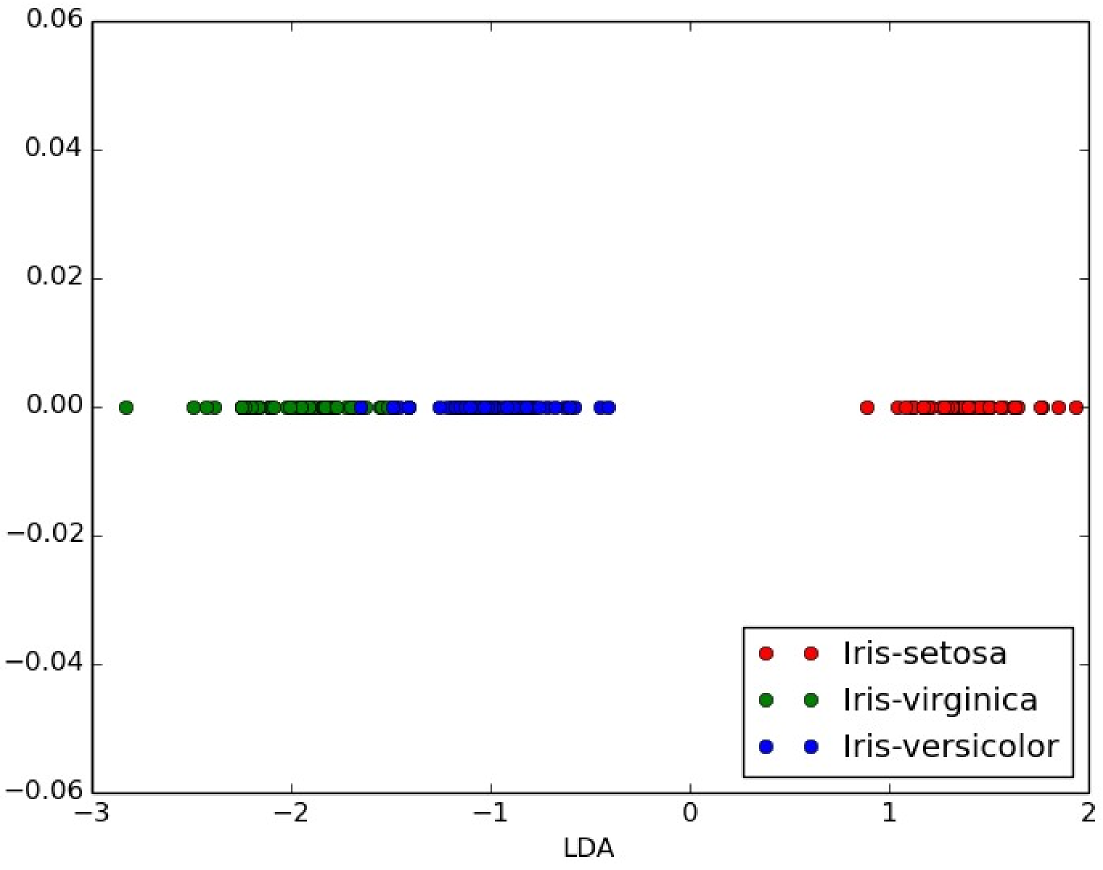
```
def calculating_scater_w(sample1, sample2, sample3):

    sample1 = np.transpose(sample1)
    scatter1 = np.dot(sample1, sample1.T)

    sample2 = np.transpose(sample2)
    scatter2 = np.dot(sample2, sample2.T)

    sample3 = np.transpose(sample3)
    scatter3 = np.dot(sample3, sample3.T)
    scatter_w = scatter1+scatter2+scatter3

    return scatter_w


def calculating_scater_b(sample, mean1, mean2, mean3, mean4):

    for i in range(0, 3):
        m[i][0]=m[i][0]-mean1
        m[i][1]=m[i][1]-mean2
        m[i][2]=m[i][2]-mean3
        m[i][3]=m[i][3]-mean4


    scatter_b=np.dot(m.T,m)
    scatter_b = scatter_b*50
    return scatter_b

def calculating_lda(scatterw, scatterb, sample):

    eigenvalue, eigenvector = np.linalg.eig(np.linalg.inv(scatterw).dot(scatterb))

    eigenpairs=[]
    for i in range(len(eigenvalue)):
        eigenpairs.append((np.abs(eigenvalue[i]), eigenvector[:,i]))

    sorted(eigenpairs,reverse=True)
    lda = np.hstack((eigenpairs[0][1].reshape(4,1)))
    temp = sample.T
    lda=np.dot(temp,lda)

    return lda



def save_plot(x1, x2, x3, name):

    plt.xlabel("LDA")
    class1=plt.plot(x1,  50*[0],'ro',label="Iris-setosa")
    class3=plt.plot(x3,  50*[0],'go',label="Iris-virginica")
```

```
        class2=plt.plot(x2,  50*[0],'bo',label="Iris-versicolor")


        location='lower right'
        plt.legend(loc=location)
        plt.savefig(name)
        plt.close()


def plot(lda):
        x1, x2, x3 = lda[0:50], lda[50:100], lda[100:150]

        save_plot(x1, x2, x3, 'lda.png')


if __name__ == '__main__':

        filename = "iris.data"
        index_of_class=4
        m=np.zeros(12)
        m=m.reshape(3,4)
        sample1, sample2, sample3, m = read_file_load_dataset(filename, index_of_class,
m)

        sample = np.concatenate([sample1, sample2, sample3])
        sample = sample.reshape(150, 4)
        sample = np.transpose(sample)

        sample1 = sample1.reshape(50, 4)
        sample2 = sample2.reshape(50, 4)
        sample3 = sample3.reshape(50, 4)


        for i in range(0, 50):
                sample1[i]=sample1[i]-m[0]
                sample2[i]=sample2[i]-m[1]
                sample3[i]=sample3[i]-m[2]

        scatter_w = calculating_scater_w(sample1, sample2, sample3)
        mean1, mean2, mean3, mean4 = np.mean(sample[0]), np.mean(sample[1]),
np.mean(sample[2]), np.mean(sample[3])
        scatter_b = calculating_scater_b(sample, mean1, mean2, mean3, mean4)

        lda= calculating_lda(scatter_w, scatter_b, sample)

        plot(lda)
```

SAMPLE OUTPUT:

Projection of the original data in PCA space

Plot for LDA in One-Dimension:



LDA on Arcene Data Set (Data Set with large Dimensions) :

LDA on Arcene data set is not possible since the with-in-class scatter matrix Sw is not invertible in this case.

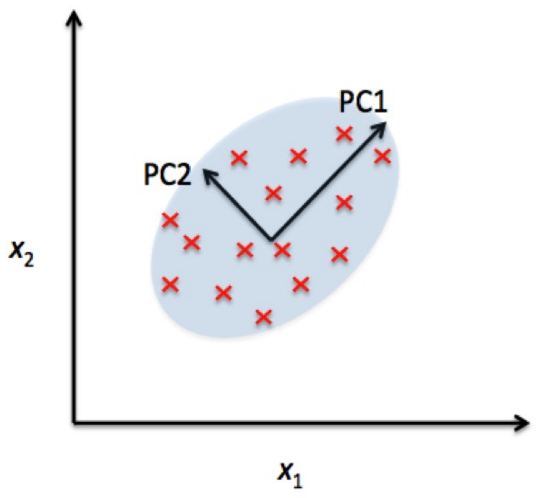
LDA vs PCA:

Similarities:

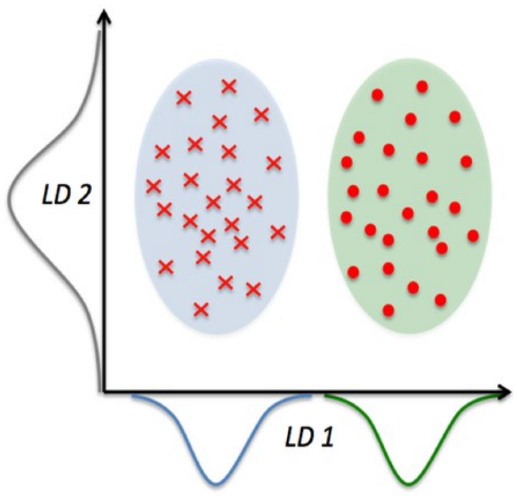
- ✔ Both LDA and PCA can be used for topic modelling.
- ✔ They are both implemented in many software packages for topic modelling.
- ✔ Both LDA and PCA are linear transformation techniques

Differences:

- ✔ PCA is a more generic dimensionality reduction technique whereas LDA is a more specialized generative method.
- ✔ LDA is a supervised whereas PCA is unsupervised (ignores class labels).
- ✔ PCA is a technique that finds the directions of maximal variance and LDA is a technique that also cares about class separability (LDA would be a very bad linear discriminant)
- ✔ LDA makes assumptions about normally distributed classes and equal class covariances.



PC1
PC2
x_2
x_1



LD 2
LD 1

B) Clustering

Iris Data Set :

```
import numpy as np
import random
import csv
import matplotlib.pyplot as plt
from tabulate import tabulate
import math


def read_file_load_dataset(filename, index_of_class):
        with open('iris.data','rb') as f:
                reader=csv.reader(f)
                sample=list(reader)
        random.shuffle(sample)
        for i in range(0, len(sample)):
                for j in range(0, len(sample[i])-1):
                        sample[i][j] = float(sample[i][j])
        for i in range(0, len(sample)):
                sample[i].extend('0')

        return sample

def calculating_distances(data, m1, m2, m3):
        dist1, dist2, dist3 =-1.0, -1.0, -1.0
        mindist=10000.0
        temp1, temp2, temp3 =m1[:-2], m2[:-2], m3[:-2]
        temp1 = np.asarray(temp1)
        temp2 = np.asarray(temp2)
        temp3 = np.asarray(temp3)

        dist1 = data-temp1
        dist2 = data-temp2
        dist3 = data-temp3

        dist1=np.dot(dist1, dist1)
        dist2=np.dot(dist2, dist2)
        dist3=np.dot(dist3, dist3)
        return dist1, dist2, dist3

def finding_min_distance(dist1, dist2, dist3, sample):
        mindist=dist1
        sample[-1]='1'
        if mindist>dist2:
                sample[-1]='2'
                mindist=dist2

        if mindist>dist3:
```

```
        sample[-1]='3'
        mindist=dist3

    return mindist, sample


def assign_clusters(l, m1, m2, m3, previousm1, previousm2, previousm3):
    no_of_iterations=0
    flag=0

    while flag!=1:
        no_of_iterations+=1
        for k in range(len(l)):
            data=l[k]
            data=data[:-2]
            data = np.asarray(data)
            dist1, dist2, dist3 = calculating_distances(data, m1, m2, m3)
            mindist, l[k] = finding_min_distance(dist1, dist2, dist3, l[k])
        print dist1, dist2, dist3
        cnt1, cnt2, cnt3 = 0,0,0
        m=np.zeros(12).reshape(3,4)
        for data in l:
            if data[-1]=='1':
                for j in range(len(data)-2):
                    m[0][j]=m[0][j]+data[j]
                cnt1+=1
            elif data[-1]=='2':
                for j in range(len(data)-2):
                    m[1][j]=m[1][j]+data[j]
                cnt2+=1
            else:
                for j in range(len(data)-2):
                    m[2][j]=m[2][j]+data[j]
                cnt3+=1
        m[0], m[1], m[2] =(m[0]/cnt1), (m[1]/cnt2), (m[2]/cnt3)
        for i in range(0, 4):
            m1[i], m2[i], m3[i] =m[0][i], m[1][i], m[2][i]
        if previousm1==m1 and previousm2==m2 and previousm3==m3:
            flag=1

        previousm1, previousm2, previousm3 =m1[:], m2[:], m3[:]
    return no_of_iterations, l

def plot(sample):
    # Plot between sepal length and petal length
    x1, x2, x3, y1, y2, y3 = [], [] ,[] , [], [], []

    for i in range(len(sample)):
        if sample[i][-1]=='1':
            x1.append(sample[i][1])
```

```
                y1.append(sample[i][3])
            elif sample[i][-1]=='2':
                x2.append(sample[i][1])
                y2.append(sample[i][3])
            else:
                x3.append(sample[i][1])
                y3.append(sample[i][3])
        plt.xlabel("Sepal Width")
        plt.ylabel("Petal Width")
        plt.plot(x1,y1,'ro',label="1")
        plt.plot(x2,y2,'bo',label="2")
        plt.plot(x3,y3,'go',label="3")
        location = "upper left"
        plt.legend(loc=location)
        plt.show()
        plt.savefig("clustering_irs.png")
        plt.close()

def confusion_matrix_calculation(actual_classes, predictions):
        #Fetching the name of the classes to dictionary and then to the list
        c = ['Iris-virginica', 'Iris-setosa', 'Iris-versicolor']
        confusion_matrix={'1':[0,0,0,0], '2':[0,0,0,0], '3':[0,0,0,0]}
        class_value=[0, 0, 0]
        for j in range(len(predictions)):
            for i in confusion_matrix.keys():
                if i==predictions[j]:
                    if actual_classes[j]==c[0]:
                        confusion_matrix[i][0]+=1
                        confusion_matrix[i][3]+=1
                    elif actual_classes[j]==c[1]:
                        confusion_matrix[i][1]+=1
                        confusion_matrix[i][3]+=1
                    elif actual_classes[j]==c[2]:
                        confusion_matrix[i][2]+=1
                        confusion_matrix[i][3]+=1

        return confusion_matrix

def external_measures(confusion_matrix):
        purity=0.0
        f_measure=0.0

        #Calculating Purity and F-Measure
        for i in confusion_matrix.keys():
            max=0.0
            for j in range(len(confusion_matrix[i])-1):
                if confusion_matrix[i][j]>max:
                    max = confusion_matrix[i][j]
                    tij   =   confusion_matrix['1'][j]   +   confusion_matrix['2'][j]   +
confusion_matrix['3'][j]
```

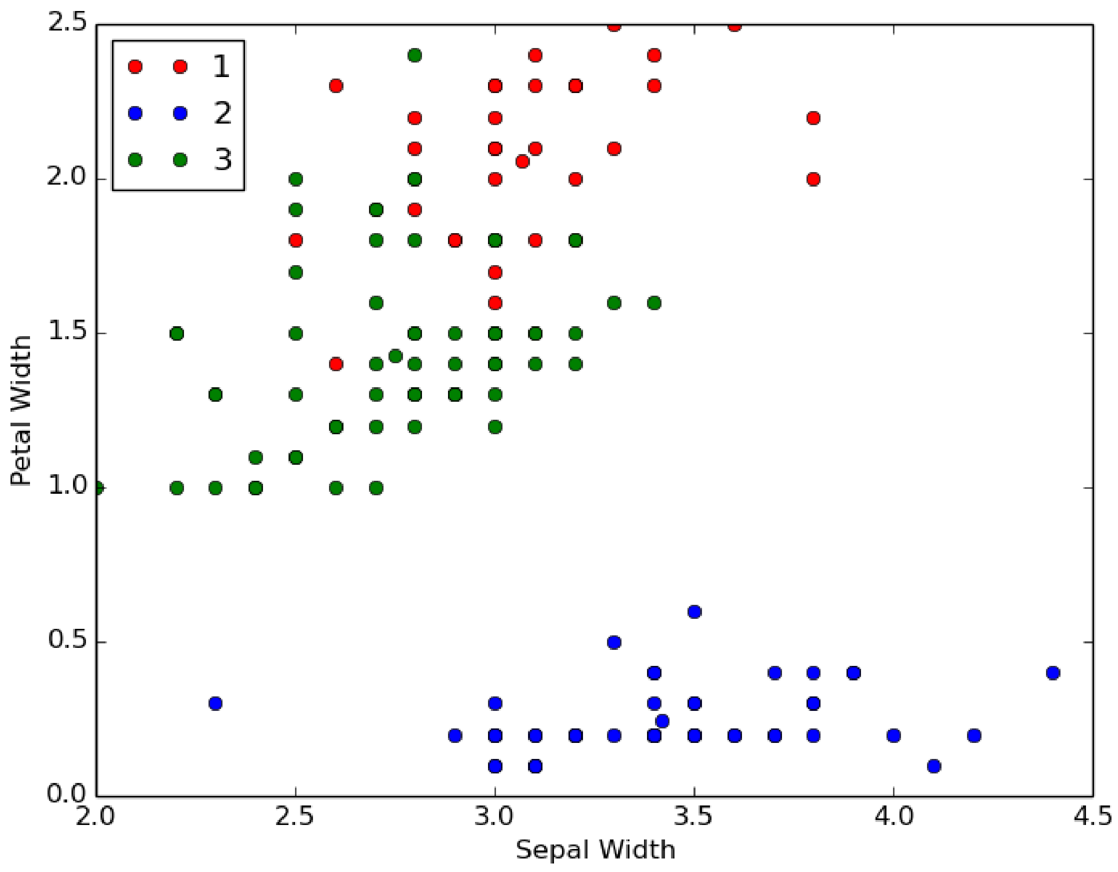
```
            purity=purity+(float(max)/float(confusion_matrix[i][3]))
            den = float(confusion_matrix[i][3])+float(tij)
            num = 2 * float(max)
            f_measure+=float(num)/float(den)

        purity = purity/3.0
        f_measure = f_measure/3.0
        return purity, f_measure

def internal_measures(confusion_matrix, sample):
        beta_cv=0.0
        normalised_cut=0.0
        w_in, w_out= 0.0, 0.0

        for datai in sample:
                for dataj in sample:
                        c1, c2 =datai[:-2], dataj[:-2]
                        c1, c2 = np.asarray(c1), np.asarray(c2)
                        c = c1-c2
                        dist=np.dot(c,c)
                        dist=math.sqrt(dist)
                        if datai[-1]==dataj[-1]:
                                w_in+=dist
                        else:
                                w_out+=dist

        w_in=w_in /2.0
        w_out=w_out/2.0

        #Finding number of distinct inter/intra cluster edges
        N_in, N_out=0.0, 0.0
        n=[]
        for i in confusion_matrix.keys():
                max_value=0.0
                for j in range(len(confusion_matrix[i])-1):
                        if confusion_matrix[i][j]>max_value:
                                max_value = confusion_matrix[i][j]
                n.append(max_value)
                temp = math.factorial(max_value)
                temp = temp/float(math.factorial(max_value-2))
                temp/=2.0
                N_in=N_in+temp
        sum=0.0
        for i in range(len(n)-1):
                sum = sum+n[i]*n[i+1]
        N_out=sum+(n[0]*n[2])
        num=float(w_in)/float(N_in)
        den=float(w_out)/float(N_out)
        beta_cv=float(num)/float(den)
        #Finding normalised cut
        normalised_cut=1/float(float(w_in)/float(w_out)+1)

        return beta_cv, normalised_cut

if __name__ == '__main__':

        filename = "iris.data"
        index_of_class=4
        sample = read_file_load_dataset(filename, index_of_class)
        sample1 = np.array(sample)
        labels = sample1[:,-1]

        #Forming Clusters - Randomly initialise K (3 here) clusters
        m1, m2, m3 = sample[0], sample[1], sample[2]
        m1[-1], m2[-1], m3[-1] = '1', '2', '3'
        previous_m1, previous_m2, previous_m3 = m1[:], m2[:], m3[:]
        random.shuffle(sample)

        #Assigning Clusters
        no_of_iterations, sample = assign_clusters(sample, m1, m2, m3, previous_m1,
previous_m2, previous_m3)
        print
        print "No of Iterations" , no_of_iterations
        print

        #Plotting
        plot(sample)

        #Confusion Matrix
        sample1 = np.array(sample)
        actual_classes = sample1[:,-2]
        predictions = sample1[:, -1]

        conf = confusion_matrix_calculation(actual_classes, predictions)

        print "Confusion Matrix"
        print conf
        print

        purity, f_measure = external_measures(conf)
        beta_cv, normalised_cut = internal_measures(conf, sample)

        print '***********External Measures***************'
        print "PURTY............: ", purity
        print "F-Measure........: ", f_measure
        print
        print '***********Internal Measures***************'
        print "Beta-CV..........: ", beta_cv
        print "Normalised Cut...: ", normalised_cut
```

```
    print '********************************************'
    print
```

SAMPLE OUTPUT:

```
14.9 0.0 16.92
10.8928385374 0.0 23.7861536009
10.9225356493 0.0 24.7773545778
10.9665898544 0.0 24.7464226966
11.0740716663 0.0 24.9329297097
11.0758704787 0.0 24.9376187864
11.0759004612 0.0 24.9377360276
11.0759009609 0.0 24.9377389586
11.0759009693 0.0 24.9377390319
11.0759009694 0.0 24.9377390337
11.0759009694 0.0 24.9377390338
11.0759009694 0.0 24.9377390338
11.0759009694 0.0 24.9377390338

No of Iterations 13

Confusion Matrix
{'1': [13, 0, 47, 60], '3': [37, 0, 3, 40], '2': [0, 50, 0, 50]}

***********External Measures***************
PURTY............:  0.902777777778
F-Measure........:  0.892255892256

***********Internal Measures***************
Beta-CV..........:  0.267581370008
Normalised Cut...:  0.879930304346
********************************************
```

Plot for Clustering for IRIS -Dataset:



CONFUSION MATRIX:

{'1': [36, 0, 4 ,40], '3': [0, 50, 0, 50], '2': [14, 0, 46, 60]}

Clusters	Iris-virginica	Iris-setosa	Iris-versicolor
1 (40)	36	0	4
2 (50)	0	50	0
3 (60)	14	0	46

ANALYSIS & DISCUSSION OF THE RESULTS-

Cluster validation measures - External Measure:

Matching Based Measure : Purity:

It quatifies the extent that cluster Ci contains points from only one partition.
Total purity of whole cluster is the sum of the purity of all the clusters divided by the number of clusters .
Purity value for IRIS Data Set: 0.902777777778
Perfect clustering is the one if purity = 1.

In our clustering method, few classes are clustered as a diffferent one and hence purity reduced to this value.

F-Measure:

F-Measure is the harmonic mean of precision and recall. Precision value is equivalent to purity.
Recall gives the fraction in point in partition shared in cluster.
F measure = 2*nij / ni + mj

Total F_measure for IRIS data set = (f_measure_1 + f_measure_2 + f_measure_3)/3
= 0.884718407758

Cluster validation measures - Internal Measure:

Beta_CV Measure:

A trade-off in maximizing intra-cluster compactiness and inter-cluster separation.
Given a clustering C = {c1, c2,....ck} with k clusters, cluster Ci, containing ni = |Ci| points.
We calculate the sum of all intra-cluster weights over all clusters.
And, we calculate the sum of all the inter-cluster weights.

The number of distinct intra-cluster edges, and inter-cluster edges are calculated.

The Beta-CV measure is the ratio of the mean intra cluster distance to the mean inter-cluster distance.

The smaller the value, the better is the clustering.
Beta-CV measure value IRIS-Data Set = 0.280803376232

Normalised Cut:

The higher the normalised cut value, the better the clustering.
Normalised Cut value for Iris Data Set : 0.875663535322

Clustring on Breast-Cancer-Wisconsin Data Set :

```
import numpy as np
import random
import csv
import matplotlib.pyplot as plt
from tabulate import tabulate
import math


def read_file_load_dataset(filename, index_of_class):
	with open(filename,'rb') as f:
		reader=csv.reader(f)
		sample=list(reader)
	count_class_2=0
	count_class_4=0
	random.shuffle(sample)
	for i in range(0, len(sample)):
		temp=[]
		for j in range(1, len(sample[i])):
			if sample[i][j]=='?':
				sample[i][j]=5
			temp.append(int(sample[i][j]))
		if sample[i][-1]=='2':
			count_class_2+=1
		if  sample[i][-1]=='4':
			count_class_4+=1
		sample[i]=temp
	for i in range(0, len(sample)):
		sample[i].extend('0')
	return sample, count_class_2, count_class_4

def calculating_distances(data, m1, m2):
	dist1, dist2=-1.0, -1.0
	mindist=10000.0
	temp1, temp2 =m1[:-2], m2[:-2]
	temp1 = np.asarray(temp1)
	temp2 = np.asarray(temp2)

	dist1 = data-temp1
	dist2 = data-temp2

	dist1=np.dot(dist1, dist1)
	dist2=np.dot(dist2, dist2)
	return dist1, dist2

def finding_min_distance(dist1, dist2,  sample):
	mindist=dist1
	sample[-1]='1'
	if mindist>dist2:
```

```
            sample[-1]='2'
            mindist=dist2

        return mindist, sample


def assign_clusters(l, m1, m2, previousm1, previousm2, count_class_2, count_class_4):
        no_of_iterations=0
        flag=0

        while flag!=1:
            no_of_iterations+=1
            for k in range(len(l)):
                data=l[k]
                data=data[:-2]
                data = np.asarray(data)
                dist1, dist2= calculating_distances(data, m1, m2)
                mindist, l[k] = finding_min_distance(dist1, dist2, l[k])
            print dist1, dist2
            cnt1, cnt2 = 0,0
            m=np.zeros(18).reshape(2,9)
            for data in l:
                if data[-1]=='1':
                    for j in range(len(data)-2):
                        m[0][j]=m[0][j]+data[j]
                    cnt1+=1
                elif data[-1]=='2':
                    for j in range(len(data)-2):
                        m[1][j]=m[1][j]+data[j]
                    cnt2+=1
            m[0], m[1] =(m[0]/cnt1), (m[1]/cnt2)
            for i in range(0, 9):
                m1[i], m2[i] =m[0][i], m[1][i]
            if previousm1==m1 and previousm2==m2:
                flag=1

            previousm1, previousm2  =m1[:], m2[:]
        return no_of_iterations, l

def confusion_matrix_calculation(actual_classes, predictions):
        #Fetching the name of the classes to dictionary and then to the list
        c = ['2', '4']
        confusion_matrix={'1':[0,0,0], '2':[0,0,0]}
        class_value=[0, 0, 0]
        for j in range(len(predictions)):
            for i in confusion_matrix.keys():
                if i==predictions[j]:
                    if actual_classes[j]==c[0]:
                        confusion_matrix[i][0]+=1
                        confusion_matrix[i][2]+=1
                        elif actual_classes[j]==c[1]:
                                confusion_matrix[i][1]+=1
                                confusion_matrix[i][2]+=1

        return confusion_matrix

def external_measures(confusion_matrix):
        purity=0.0
        f_measure=0.0

        #Calculating Purity and F-Measure
        for i in confusion_matrix.keys():
                max=0.0
                for j in range(len(confusion_matrix[i])-1):
                        if confusion_matrix[i][j]>max:
                                max = confusion_matrix[i][j]
                                tij = confusion_matrix['1'][j] + confusion_matrix['2'][j]
                purity=purity+(float(max)/float(confusion_matrix[i][2]))
                den = float(confusion_matrix[i][2])+float(tij)
                num = 2 * float(max)
                f_measure+=float(num)/float(den)

        purity = purity/2.0
        f_measure = f_measure/2.0
        return purity, f_measure

def internal_measures(confusion_matrix, sample):
        beta_cv=0.0
        normalised_cut=0.0
        w_in, w_out= 0.0, 0.0

        for datai in sample:
                for dataj in sample:
                        c1, c2 =datai[:-2], dataj[:-2]
                        c1, c2 = np.asarray(c1), np.asarray(c2)
                        c = c1-c2
                        dist=np.dot(c,c)
                        dist=math.sqrt(dist)
                        if datai[-1]==dataj[-1]:
                                w_in+=dist
                        else:
                                w_out+=dist

        w_in=w_in /2.0
        w_out=w_out/2.0


        #Finding number of distinct inter/intra cluster edges
        N_in, N_out=0.0, 0.0
        n=[]
```

```
	for i in confusion_matrix.keys():
		max_value=0.0
		for j in range(len(confusion_matrix[i])-1):
			if confusion_matrix[i][j]>max_value:
				max_value = confusion_matrix[i][j]
		n.append(max_value)
		temp = math.factorial(max_value)
		temp = temp/math.factorial(max_value-2)
		temp=temp/2.0
		N_in=N_in+temp
	N_out=(n[0]*n[1])
	num=float(w_in)/float(N_in)
	den=float(w_out)/float(N_out)
	beta_cv=float(num)/float(den)

	#Finding normalised cut
	normalised_cut=1/float(float(w_in)/float(w_out)+1)

	return beta_cv, normalised_cut

if __name__ == '__main__':

	filename = "breast-cancer-wisconsin.data"
	index_of_class=4
	sample, count_class_2, count_class_4 = read_file_load_dataset(filename,
index_of_class)

	#Forming Clusters - Randomly initialise K (3 here) clusters
	m1, m2 = sample[0], sample[1]
	m1[-1], m2[-1]= '1', '2'
	previous_m1, previous_m2 = m1[:], m2[:]
	random.shuffle(sample)
	sample1 = np.array(sample)
	#Assigning Clusters
	no_of_iterations, sample = assign_clusters(sample, m1, m2, previous_m1,
previous_m2, count_class_2, count_class_4)

	print
	print "No of Iterations" , no_of_iterations
	print

	#Confusion Matrix
	sample2 = np.array(sample)
	actual_classes = sample1[:,-2]
	predictions = sample2[:, -1]

	conf = confusion_matrix_calculation(actual_classes, predictions)

	print "Confusion Matrix"
	print conf
```

```
    print

    purity, f_measure = external_measures(conf)
    beta_cv, normalised_cut = internal_measures(conf, sample)

    print '***********External Measures**************'
    print "PURTY............: ", purity
    print "F-Measure........: ", f_measure
    print
    print '***********Internal Measures**************'
    print "Beta-CV..........: ", beta_cv
    print "Normalised Cut...: ", normalised_cut
    print '******************************************'
    print
```

SAMPLE OUTPUT:

```
13 255
5.2872383758 224.949806768
4.77491827725 213.243942524
4.53596651126 205.926446456
4.47256699606 204.337013434
4.47243358863 204.33024328
4.47243330175 204.330214348
4.47243330113 204.330214224
4.47243330113 204.330214224
4.47243330113 204.330214224
4.47243330113 204.330214224

No of Iterations 11

Confusion Matrix
{'1': [447, 18, 465], '2': [11, 223, 234]}

***********External Measures**************
PURTY.............:  0.957140887786
F-Measure........:  0.95376404174

***********Internal Measures**************
Beta-CV...............:  0.331704165291
Normalised Cut...:  0.707176741294
******************************************
```

CONFUSION MATRIX:

{'1': [447, 18, 465], '2': [11, 223, 234]}

Clusters	'2'	'4'
1 (465)	447	18
2 (234)	11	223

ANALYSIS & DISCUSSION OF THE RESULTS-

Cluster validation measures - External Measure:

Matching Based Measure : Purity:

Purity value for Wisconsin Data Set: 0.957140887786
Perfect clustering is the one if purity = 1.
In our clustering method, few classes are clustered as a diffferent one and hence purity reduced to this value.

F-Measure:

Total F_measure for Wisconsin data set = (f_measure_1 + f_measure_2 /2
= 0.95376404174

Cluster validation measures - Internal Measure:

Beta_CV Measure:

Beta-CV measure value Wisconsin-Data Set = 0.331704165291

Normalised Cut:

Normalised Cut value for Wisconsin Data Set : 0.707176741294

QUESTIONS TO BE ANSWERED:

Q-1 Compare and contrast K-Means, K-median and K-medoid approaches – when do you use each method, what are the differences in the objective function that is optimized in each case.

K-Means

k-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.
k-means minimizes within-cluster variance, which equals squared Euclidean distances. In general, the arithmetic mean does this. It does not optimize distances, but squared deviations from the mean.

* If the distance is **squared Euclidean distance**, we use **k-means.**

K-Median

k-medians clustering is a cluster analysis algorithm. It is a variation of *k*-means clustering where instead of calculating the mean for each cluster to determine its centroid, one instead calculates the median. This has the effect of minimizing error over all clusters with respect to the 1-norm distance metric, as opposed to the square of the 2-norm distance metric (which *k*-means does.)
k-medians minimizes absolute deviations, which equals Manhattan distance. In general, the median does this. It is a good estimator for the mean, if you want to minimize absolute deviations, instead of squared ones.

*If the distance is **squared Euclidean distance**, we use **k-means.**

K-Medoid

The *k*-medoids algorithm is a clustering algorithm related to the *k*-means algorithm and the medoidshift algorithm. Both the *k*-means and *k*-medoids algorithms are partitional (breaking the dataset up into groups) and both attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the *k*-means algorithm, *k*-medoids chooses datapoints as centers (medoids or exemplars).
It could be more robust to noise and outliers as compared to k-means because it minimizes a sum of general pairwise dissimilarities instead of a sum of squared Euclidean distances. The possible choice of the dissimilarity function is very rich but in our applet we used the squared Euclidean distance.

* If we have **any other distance,** we use **k-medoids.**

Q-2 What is the difference between using Covariance matrix versus Correlation matrix for doing PCA on data? When do you recommend using one over the other? Demonstrate this on a small dataset to support your argument.

Ans.
The classic approach to PCA is to perform the eigendecomposition on the covariance matrix Σ, which is a d×d matrix where each element represents the covariance between two features. The covariance between two features is calculated as follows:

$$\sigma jk=1/n-1\sum Ni=1(xij-\bar{x}j)(xik-\bar{x}k).$$

We can summarize the calculation of the covariance matrix via the following matrix equation:

$$\Sigma=1/n-1((X-\bar{x})T(X-\bar{x})) \text{ where } \bar{x} \text{ is the mean vector.}$$

The correlation matrix can be understood as the normalized covariance matrix. Using the correlation matrix standardises(i.e. not just centered but also rescaled) the data. In general they give different results. Especially when the scales are different.

USAGE:

- We would prefer to do PCA on correlations instead of doing it on covariance when one wants the analysis to reflect just and only linear associations.
- We would prefer to do PCA on correlations instead of doing it on covariances when one wants the associations to reflect relative co-deviatedness (from the mean) rather than raw co-deviatedness. The correlation is based on distributions, their spreads, while the covariance is based on the original measurement scale.
- We use covariance matrix when the variable scales are similar and the correlation matrix when variables are on different scales.
- PCA on correlation is much more informative and reveals some structure in the data and relationships between variable Example : For iris dataset, both corealtion and covariance matric result in same set of eigen value, eigen vector pair.

Q-3 Show the process of kernelizing PCA. What kind of dimensionality reduction does kernel-PCA accomplish?

Kernel Principal Component Analysis (Kernal PCA):

Kernel PCA with linear kernel is exactly equivalent to the standard PCA. Kernel principal component analysis (kernel PCA) is an extension of principal component analysis (PCA) using techniques of kernel methods. Using a kernel, the originally linear operations of PCA are done in a reproducing kernel Hilbert space with a non-lnear mapping.

Standard PCA only allows linear dimensionality reduction. However, if the data has more complicated structures which cannot be well represented in a linear subspace, standard PCA will not be very helpful. Fortunately, kernel PCA allows us to generalize standard PCA to nonlinear dimensionality reduction.

- ✔ Construct the kernel matrix K from the training data set
- ✔ Compute the Gram matrix G
- ✔ Solve for vector Ai
- ✔ Compute the kernel principal components Yk(x)