assignment2

chandan u

2/9/2017

library

```
#install.packages("mlbench")
```

```
library("mlbench")
data("Ionosphere")
#summary(Ionosphere)
```

Question 1

compute the co-variance for each class (good,bad): Also we will remove the first two columns as they have bollean values and zeros which will cause singular matrix errors

```
# compute the co-variance for each class (good,bad)
X = as.matrix(Ionosphere)
X_good = X[ X[,35] == "good", ]
X_good = X_good[,3:34]
X_bad = X[ X[,35] == "bad", ]
X_bad = X_bad[,3:34]

## convert the character matrix to numeric
X_good = apply(X_good,c(1,2), as.numeric)
X_bad = apply(X_bad,c(1,2), as.numeric)

## scale
X_good_scaled = scale(X_good,center=TRUE,scale=FALSE)
X_bad_scaled = scale(X_bad,center=TRUE,scale=FALSE)

## covaraince
n_good = nrow(X_good)
n_bad = nrow(X_bad)
cov_good = t(X_good_scaled) %*% X_good_scaled/n_good
cov_bad = t(X_bad_scaled) %*% X_bad_scaled/n_bad
```

The mean vector for each class (good, bad):

```
mean_good  = colMeans(X_good)
mean_bad = colMeans(X_bad)
```

Calcluating mahalanobis distance from each data point to every other data point!

```
#mahalanobis(X_good[1,], mean_good ,cov_good)

nearest = c()
samples = c(rep(1:nrow(X)))
for( row in samples){
  library('MASS')
```

```
  point = X[row, 3:34]
  point = sapply(point, as.numeric)
  dist_from_good = sqrt(t( point - mean_good) %*% solve(cov_good) %*% ( point - mean_good))
  dist_from_bad  = sqrt(t(point - mean_bad) %*% solve(cov_bad) %*% (point - mean_bad))

  if (dist_from_bad > dist_from_good) {
    nearest = c(nearest, "good")
  }
  else{
    nearest = c(nearest, "bad")
  }
}
```

Now lets compute the **confusion matrix** using the observed and predicted:

```
confMatrix = table(nearest, Ionosphere$Class)
confMatrix
```

```
##
## nearest bad good
##    bad  126  130
##    good   0   95
```

```
#colnames(confMatrix)
#rownames(confMatrix)
```

2

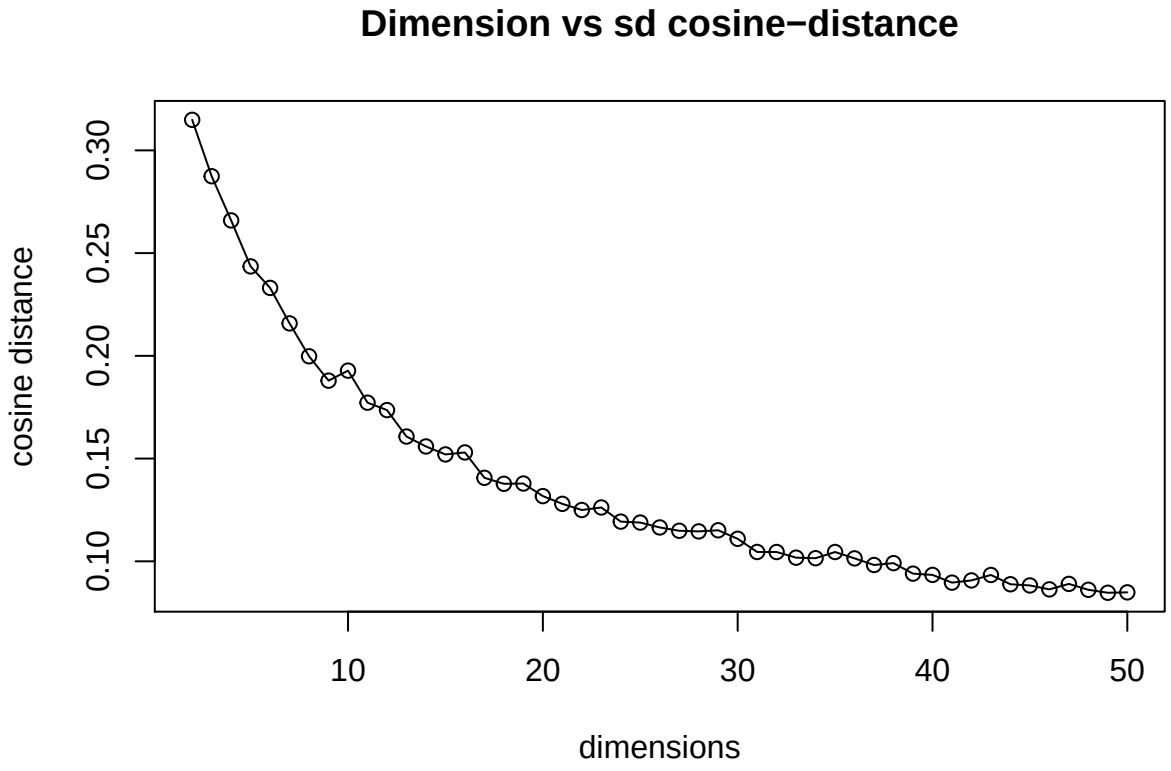
a The plots

```
## for each dimension:

number_of_points = 1000
dimensions = c(rep(2:50));

# sd and mean vectors:

cos.sd = c()
cos.mean = c()

cos.sim <- function(vector1,vector2){
  distance =  crossprod(vector1, vector2)/sqrt(crossprod(vector1) * crossprod(vector2))
  return(abs(distance))
}

computeMatrix <- function(dimension){

  # genereate values from standard normal dist
  # for each dimension
  m1 = matrix(rnorm(dimension * number_of_points), nrow=number_of_points, ncol=dimension )
  m2= matrix(rnorm(dimension * number_of_points), nrow = number_of_points, ncol = dimension)

  # list to storte cosine distances
```

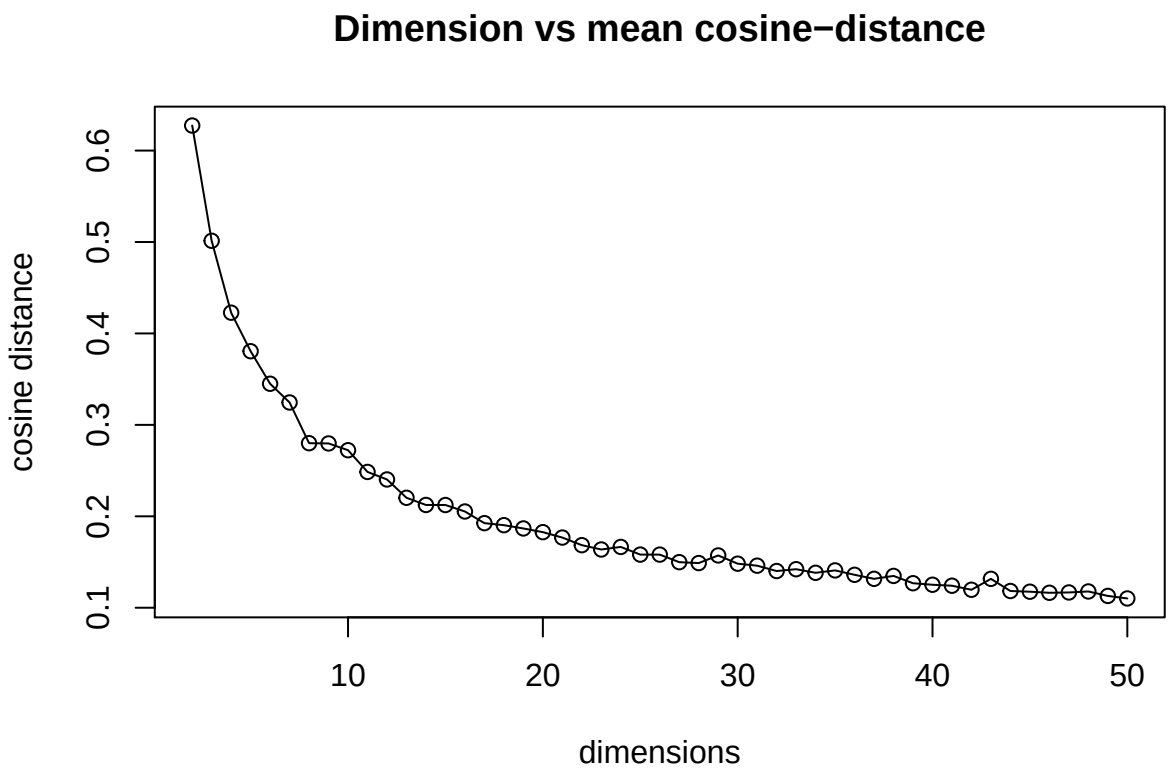
```
  # cos.matrix = matrix(data=NA,nrow= number_of_points,ncol=number_of_points)
  cos.list = c()


  # compute cosine dist between all the points
  # cosine_distances = matrix(apply(m,1, cos.sim, vector),      nrow=number_of_points, ncol=number_of_p
  for(i in c(1:number_of_points)){
    vector1 = m1[i,]
    vector2 = m2[i,]
    cos.list = c(cos.list, cos.sim(vector1, vector2))
  }#for

  # compute the sd and mean of cos.dist obtained and save in vector
  # <<- is used for accessing global variables.
  #current_length = length(cos.sd)
  cos.sd <<- append(cos.sd, sd(cos.list), after = length(cos.sd))
  cos.mean <<- append(cos.mean, mean(cos.list), after = length(cos.mean))

} #def computeMatrix


# main call dimensions: 2: 1000 execute fun::computeMatrix
s = sapply(dimensions,computeMatrix)
```

Now lets draw a plot between of mean,sd as a function of dimension [2,1000]

```
# plot mean, sd as function of dimension: [2, 1000]
plot( dimensions, cos.sd, xlab = "dimensions", ylab="cosine distance", main="Dimension vs sd cosine-dis
lines(dimensions, cos.sd)
```



```r
plot( dimensions, cos.mean, xlab = "dimensions", ylab="cosine distance", main="Dimension vs mean cosine
lines(dimensions, cos.mean)
```



b

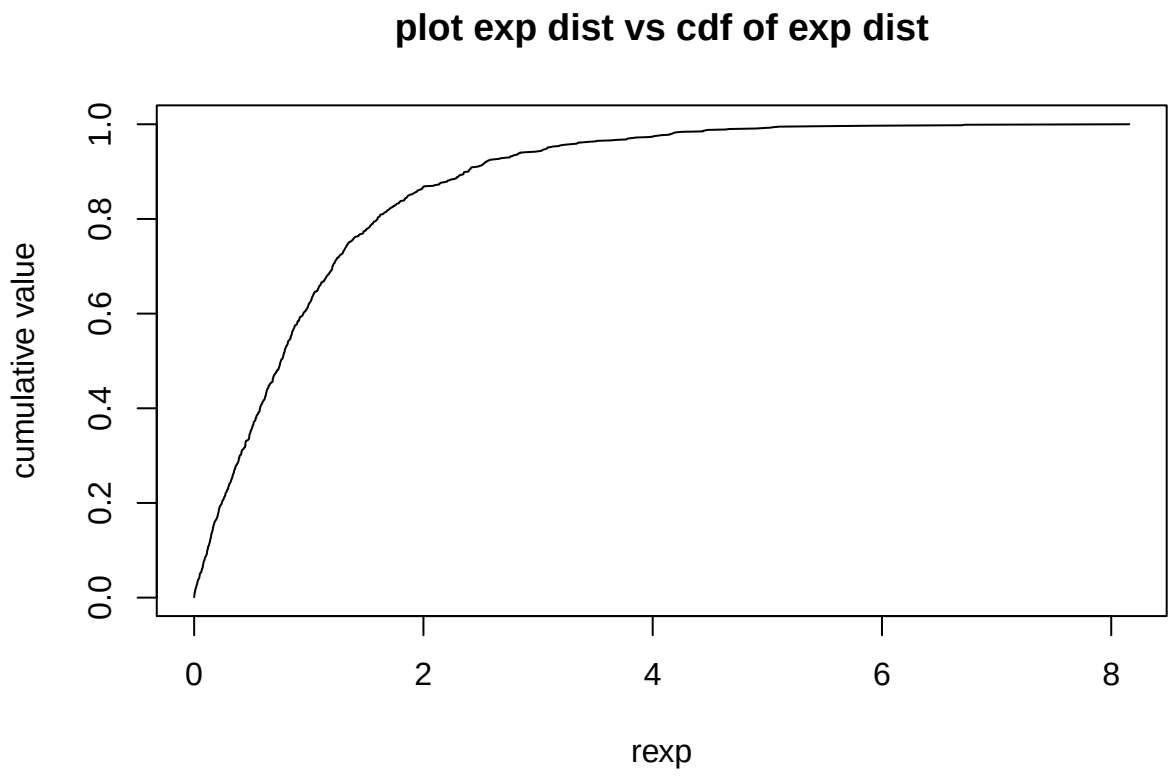
As the number of dimensions increases the mean and the sd of cosine distance between points decreases. As the dimensions increases the points tend to get closer to the edges. So the points start getting clustered at edges and hence the distance between them is reduced.

3

a

```r
# sample size
n = 1000

# random sample from exponential dist
x = rexp(1000, rate=1)


# sort values
x = sort(x)

# frequency distribution of x
x.freq = table(x)

# cdf
```

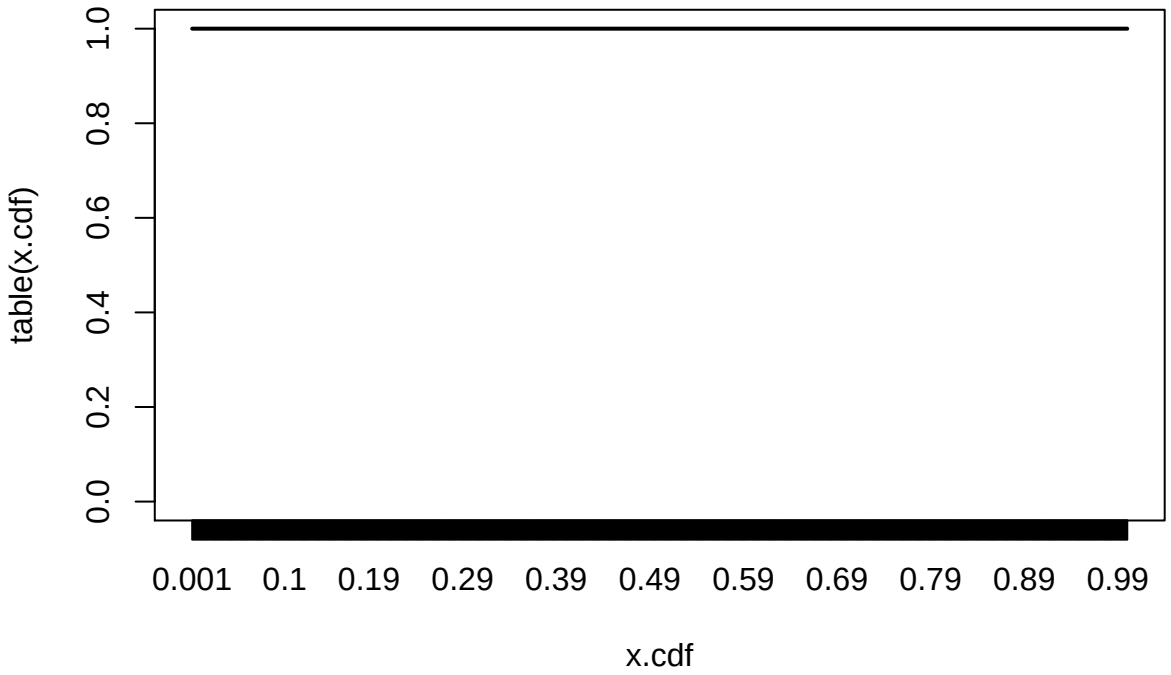
```r
x.cdf = c()
for(value in x){
  x.cdf = append(x.cdf, length(x[ x <= value])/n, after= length(x.cdf))
}

plot(x,x.cdf, type="l", xlab="rexp", ylab="cumulative value", main="plot exp dist vs cdf of exp dist")
```



b

now that we have the cdf distribution this will be our new x.lets observe the distribution of these values:

```r
plot(table(x.cdf), type="l")
```



As you can cleary see from the plot below the distribution is uniform,this is because cdf is a percentile function and it is always increasing with x. So no two cdf values are ever same.

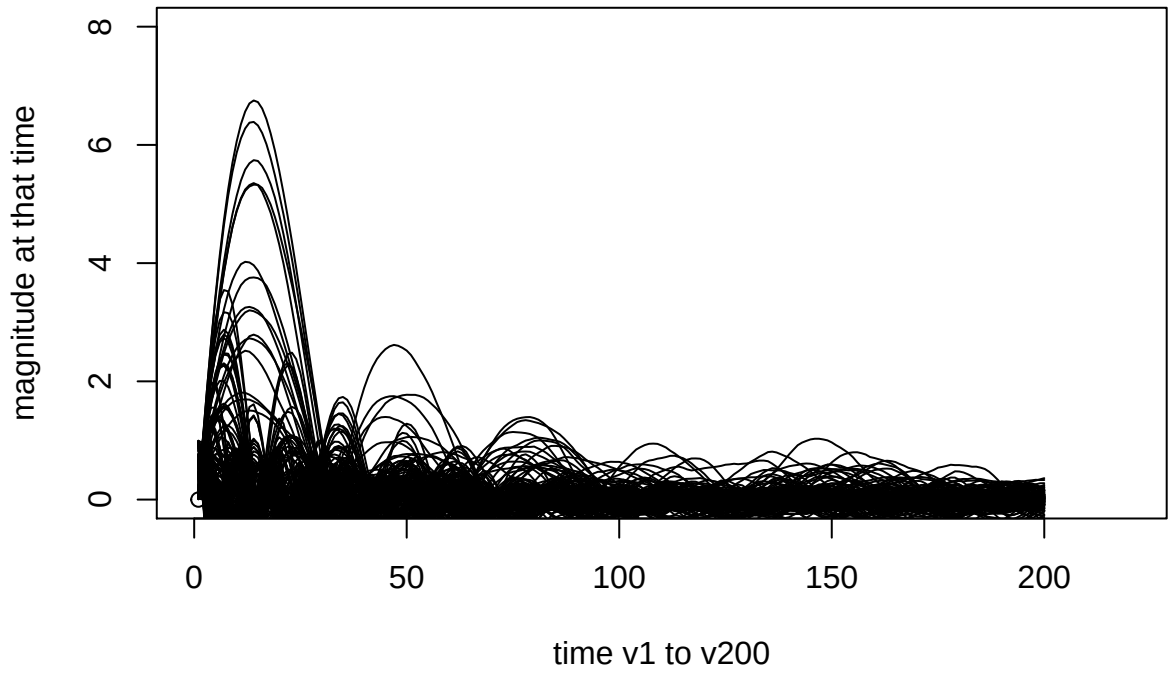
c

even if we use a different distribution and pick samples randomly, the distribution of the Cumulative distribution function values will be always be discrete uniform distribution as CDF(x) is an increasing function, and right continuous.

Question 4

Load data:

```
time_series = read.csv("../time_series.csv")

plot(0, xlim=c(0,220), ylim= c(0,8), xlab="time v1 to v200", ylab="magnitude at that time")
s=apply(time_series, 1, lines, type="l")
```



The above diagram shows the 200 time series objects. One thing is each object wave. And any wave has some properties such as amplitude and frequeny, wavelength etc. We can see that these waves differ in these properties. But for better visualization of the four models in this time-series data we have to do dimensionality reduction so we can view it better.

Since its really hard to visualize this: we will use dimensionality reductoin technique(PCA):

THE Below pca model clearly shows the four different time series models that are present in the data. It clearly catches the variance in the frequency and amplitude of these models.

```
time_series_scaled = scale(t(time_series),center=TRUE,scale=FALSE)

# compute the covariace matrix (X-u)(X-u)^T   (n*1)(1*n) matrix = n*n(matrix)
n = nrow(time_series_scaled)
cov = t(time_series_scaled) %*% time_series_scaled/n

#Perform the PCA by computing the SVD of the sample co-variance matrix:
# singular value decomposition
svd =  svd(cov) # take the singular value decomposition S = UDU^t
svd$d
```

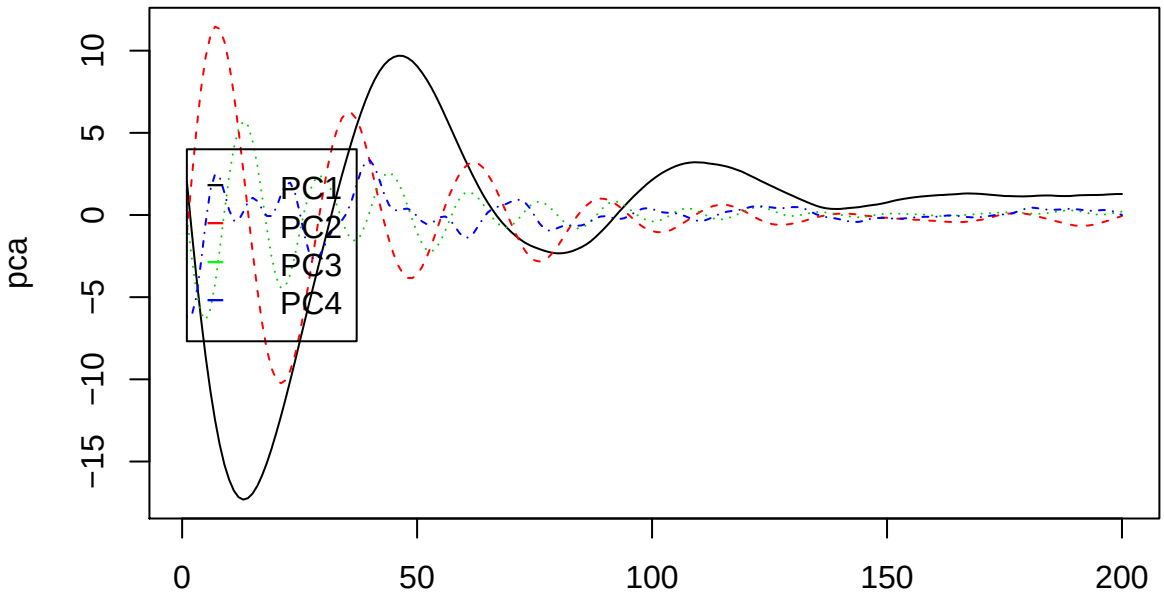
```
##   [1] 3.088140e+01 1.075793e+01 2.421150e+00 1.209324e+00 8.870475e-01
##   [6] 8.374672e-01 7.878538e-01 6.564587e-01 6.345993e-01 4.076373e-01
##  [11] 3.364172e-01 2.631405e-01 2.246526e-01 1.903228e-01 1.593088e-01
##  [16] 1.270979e-01 1.058663e-01 9.648342e-02 9.151302e-02 7.266017e-02
##  [21] 6.023790e-02 4.986990e-02 4.766949e-02 3.688749e-02 3.169013e-02
##  [26] 2.588889e-02 2.516976e-02 2.155535e-02 2.009807e-02 1.909809e-02
##  [31] 1.333814e-02 1.219439e-02 1.167837e-02 8.607370e-03 7.417253e-03
##  [36] 5.832722e-03 5.430221e-03 5.008438e-03 4.660061e-03 4.137571e-03
##  [41] 3.406939e-03 3.177639e-03 3.056740e-03 2.671985e-03 2.494235e-03
##  [46] 2.147594e-03 2.080381e-03 1.832621e-03 1.544025e-03 1.519938e-03
##  [51] 1.469829e-03 1.350174e-03 1.325464e-03 1.146811e-03 1.005121e-03
##  [56] 7.622713e-04 5.581540e-04 4.640269e-04 3.982855e-04 3.750153e-04
##  [61] 3.729164e-04 2.587465e-04 2.455107e-04 2.037037e-04 1.949404e-04
##  [66] 1.754805e-04 1.561317e-04 1.453138e-04 1.395650e-04 1.199145e-04
```

```
##  [71] 1.024463e-04 9.128698e-05 8.898158e-05 8.262899e-05 7.330978e-05
##  [76] 7.072088e-05 6.930081e-05 6.644197e-05 6.155959e-05 6.024738e-05
##  [81] 5.497421e-05 5.129101e-05 4.718300e-05 4.595671e-05 4.329402e-05
##  [86] 3.939666e-05 3.826908e-05 3.482891e-05 3.474284e-05 3.369520e-05
##  [91] 3.230512e-05 3.152766e-05 3.025055e-05 2.827517e-05 2.767592e-05
##  [96] 2.635134e-05 2.556928e-05 2.463835e-05 2.448429e-05 2.230553e-05
## [101] 2.214018e-05 2.062140e-05 1.941697e-05 1.936116e-05 1.816959e-05
## [106] 1.756904e-05 1.741948e-05 1.660062e-05 1.642303e-05 1.560801e-05
## [111] 1.475144e-05 1.418118e-05 1.348552e-05 1.325673e-05 1.290967e-05
## [116] 1.168108e-05 1.140638e-05 1.134188e-05 1.085925e-05 1.030303e-05
## [121] 1.020380e-05 9.656727e-06 9.468957e-06 8.974847e-06 8.754211e-06
## [126] 8.707321e-06 8.357979e-06 8.045750e-06 7.609415e-06 7.371200e-06
## [131] 7.109148e-06 7.004964e-06 6.787485e-06 6.477843e-06 6.342508e-06
## [136] 5.991232e-06 5.936087e-06 5.729568e-06 5.432901e-06 4.961267e-06
## [141] 4.805021e-06 4.543331e-06 4.279553e-06 4.097980e-06 4.017030e-06
## [146] 3.866974e-06 3.655003e-06 3.579279e-06 3.489975e-06 3.286965e-06
## [151] 3.094173e-06 2.926064e-06 2.680190e-06 2.603934e-06 2.557901e-06
## [156] 2.521068e-06 2.480288e-06 2.307091e-06 2.205853e-06 1.996204e-06
## [161] 1.935932e-06 1.837225e-06 1.652646e-06 1.601045e-06 1.563117e-06
## [166] 1.473572e-06 1.421814e-06 1.267627e-06 1.242490e-06 1.160848e-06
## [171] 1.095015e-06 1.007775e-06 8.607312e-07 7.756413e-07 7.370987e-07
## [176] 6.416798e-07 6.160669e-07 5.681204e-07 5.021262e-07 4.784600e-07
## [181] 4.516886e-07 4.195645e-07 3.340217e-07 2.878661e-07 2.628299e-07
## [186] 2.452781e-07 2.153111e-07 2.076508e-07 1.803210e-07 1.607580e-07
## [191] 1.347441e-07 1.006097e-07 7.906288e-08 4.935592e-08 2.364556e-08
## [196] 1.955887e-08 1.124080e-08 9.469117e-09 3.396530e-09 1.369238e-14
```

```
#Now lets perform the PCA:
pca = time_series_scaled %*% svd$u[,1:4]

# plot the principal components:
dim(pca)
```

```
## [1] 200   4
```

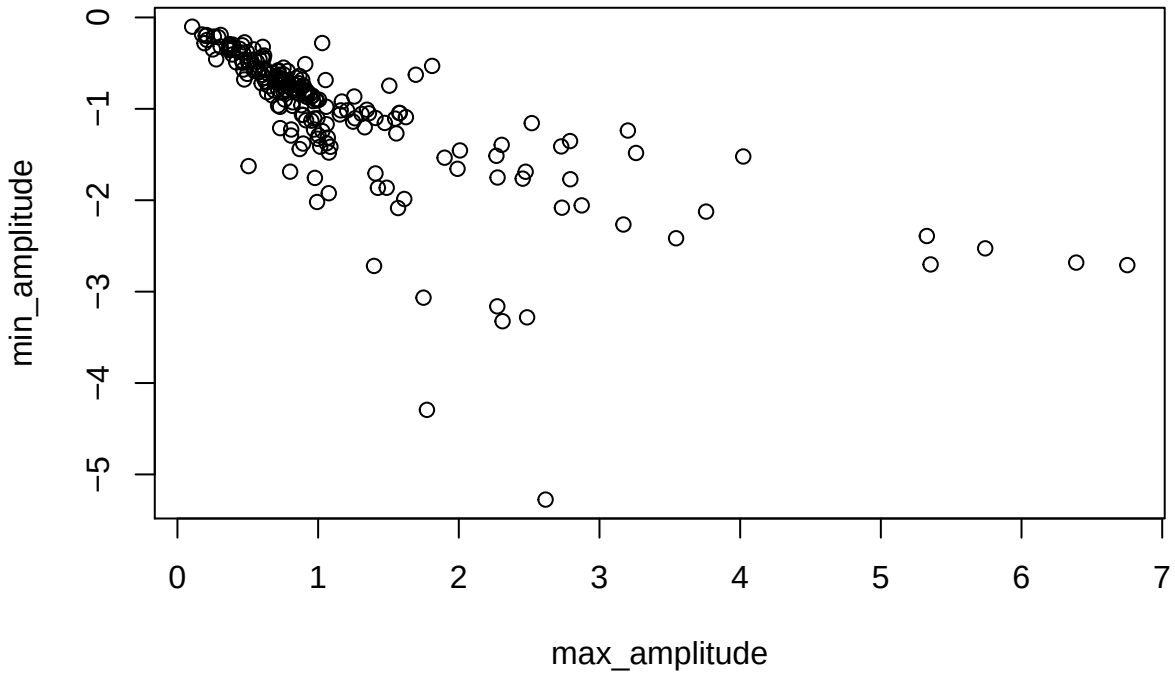
```
matplot(pca, type="l")
legend(1, 4, c("    PC1", "    PC2", "    PC3", "    PC4"), pch="----", col = c("black", "red", "green"
```



b

The two features that effectivly seperate the time series into four categories are: Maximum-amplitude(crest) and maximum Amplitude(trough) of the each time-series wave.

```
max_amplitude = c()
min_amplitude = c()
# lets derive these feautures
for(i in c(1:nrow(time_series))){
  obj = time_series[i,]
  max_amplitude <<- append(max_amplitude, max(time_series[i,]), after = length(max_amplitude))
  min_amplitude <<- append(min_amplitude, min(time_series[i,]), after = length(min_amplitude))
}

# now we have two features of each time-series object. Lets plot
plot(max_amplitude, min_amplitude)
```

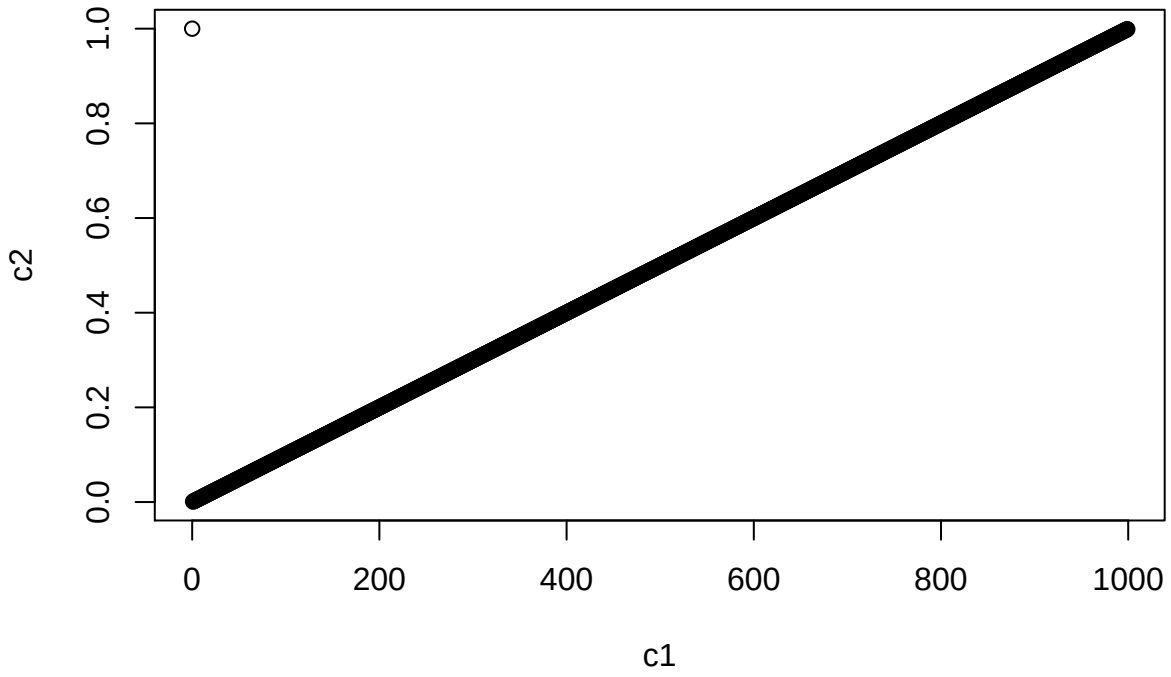


Question 5

a

Lets generate such a sample set that follows the percentile/CDF approach where one critic is postiive than other:

```
c2 = c(1:1000)/1000
c1 = c(1:999) + 1/1000
c1 = append(c1, 1/1000, after=length(c2))

plot(c1, c2)
```



b

Even though first ciritiq is more positive than second critic, neither of the critiqs ratings are dependent on each other. There is no evidence that the one critic's ratings is influenced by other.

c

It is not possible for one critic to always give a higer percentile score. There will be one less value always.

```
c2 = c(1:1000)/1000
c1 = c(1:999)/1000 + 1/1000
c1 = append(c1, 1/1000, after=length(c1))
```

d

Nope the average of the differences $x_i - y_i$ is always zero. Lets take the above critic's reviews and compute this average of differences: As you can clearly see the value is aproximately equal to zero.

```
#mean()
sum(c1-c2)/1000
```

```
## [1] 8.75168e-19
```

e

Yes Is is possible that $x_i > y_i$ for all but 1 movie. For example in the above distribution the number of points in c1 $<$ c2 are:

```
length(c1[c1 < c2])
```

```
## [1] 1
```

Questoin 6

a

It's given matrix D is with positive values. And D_{ij} represents the euclidian distance between two points $(x_i, y_i) and (x_j, y_j)$. But if i=j i.e $D_{ij} = D_{ii} = D_{jj} = 0$. i.e the distance between same points is always zero. Hence in the matrix D the diagonal elements are always zero. **And zero is neither positive nor negitive**. So it is not possible to have a matrix D that has all positive numbers. So it is impossible to find the collection of points that would give us a matrix D of positive values.

b

One way to do this is :

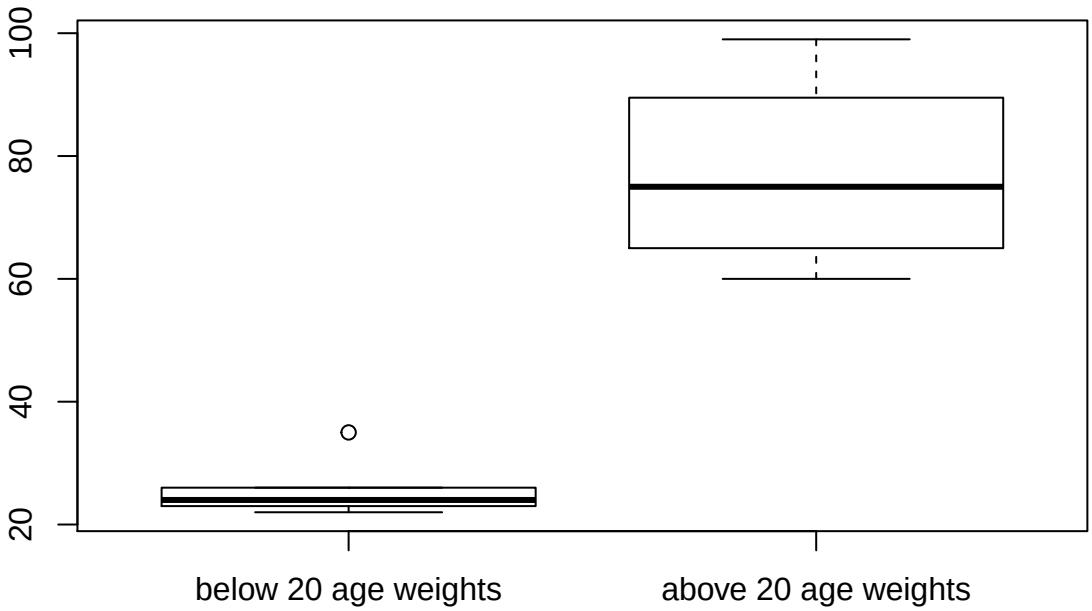
1. consider one of the points p1 as (0,0).
2. Take a p2 which is at a distance D_{12}. Use this as radius to draw a circle with center as p1 (0,0).
3. Now take a third point p3. Draw a circle with p1 as center and radius of D_{13}.
4. Now we have two circles with center as p1 (0,0) and radius as D_{12} and D_{13}.
5. Take any point on the circe with radius D_{12} and consider it as point p2.
6. We know the distance between p2 and p3 is D_{23}. From p2 draw one more circle with a radius of D_{23}. This circle intersects with the circle with center p1 and radius d_{13}. It intersects at two points. You can choose one of these points as p3.
7. We keep following the same steps for other points as well.

Question 7

Boxplots in general reveal a lot of information: 1. median 2. inter quartile range(50 percent population) 3. min 4. max 5. lower quartile, upper quartile 6. Spread of the data

So parallel boxplots could reveal intresting insights. For example lets say ur comparing weights of persons whose age is above 20 and weights of persons whose age is below 20. You build seperate box plots for each category side by side and see how the distribution is:

```
weights_below_20= c(22,23,24, 35, 26)
weights_above_20 = c(60, 80, 70, 99)

boxplot(weights_below_20, weights_above_20, names=c("below 20 age weights", "above 20 age weights"))
```



There are many ways we can use boxplots to compare attributes at various levels. We can also compare various samples.