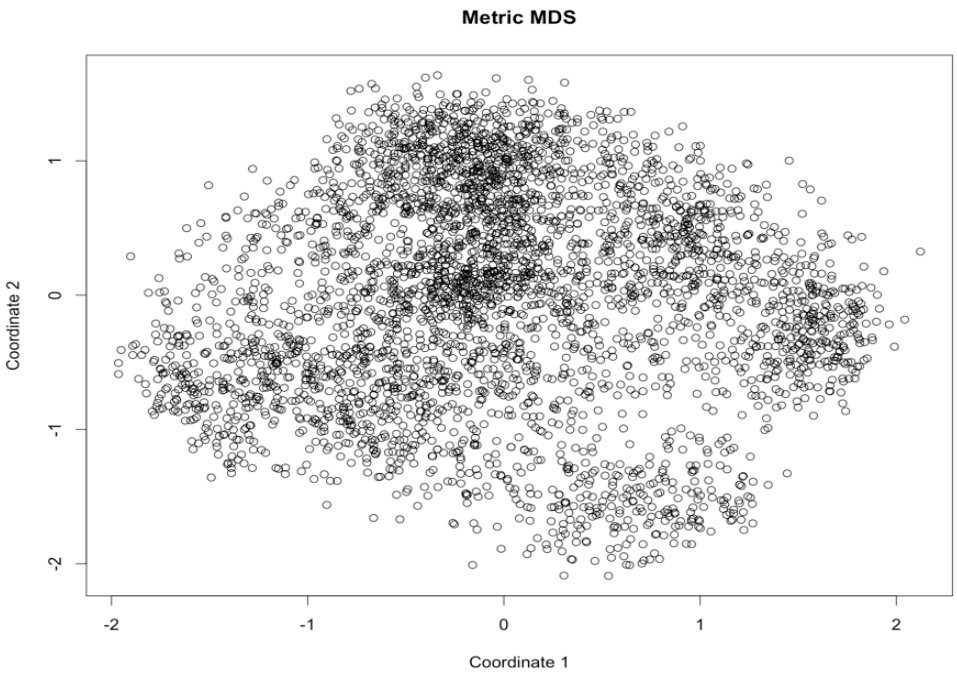
Section A: Clustering
1. K-Means Clustering:

optdigits <- read.csv("optdigits.tra")

Normalizing the data:

library(amap)

Using MultiDimensional Scaling (MSD) to plot the data
d <- dist(mydata) # euclidean distances between the rows
fit <- cmdscale(d,eig=TRUE, k=2) # k is the number of dim
fit # view results



ptm <- proc.time()
kfit <- kmeans(mydata, 10, iter.max = 100)
proc.time() – ptm

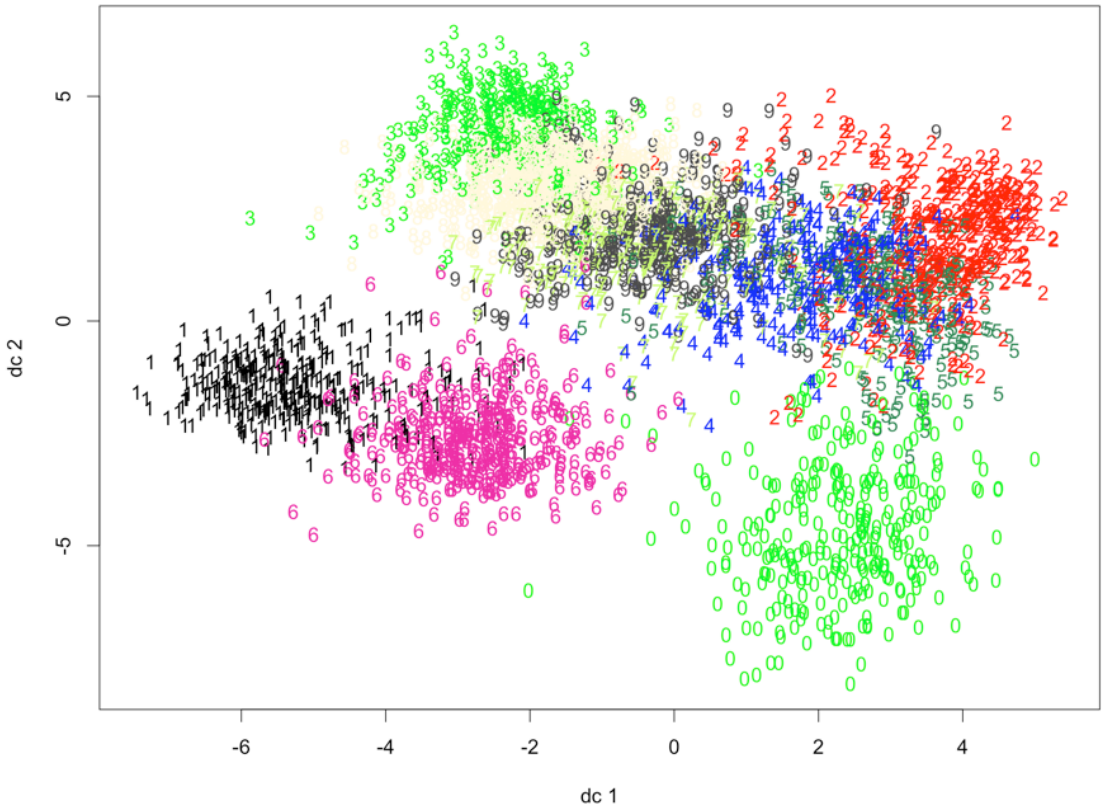
K	Time Elapse	between_SS / total_SS
2	0.018	11.3 %
3	0.051	17.8 %
4	0.045	25.4 %
5	0.042	30.3 %
6	0.042	35.7 %

7	0.037	36.5%
8	0.048	41.0 %
9	0.049	42.1 %
10	0.057	46.0 %
11	0.047	46.8 %
12	0.050	48.7 %

As increase the K, we can see that the between_SS / total_SS is increasing. And after K=10, the speed of increasing is decrease. That because the original data has 10 clusters. Next we use k=10 to do the cluster analysis.

Using MultiDimensional Scaling (MSD) to plot the cluster result:

```
library(fpc)
plotcluster(mydata, kfit$cluster)
```



Calculate the Purity:

```
ClusterPurity <- function(clusters, classes) {
    sum(apply(table(classes, clusters), 2, max)) / length(clusters)
 }

 ClusterPurity(kfit$cluster, optdigits[,65])
```

[1] 0.8082156
library(SNFtool)
calNMI(kfit$cluster, optdigits[,65])
[1] 0.7665318
library(flexclust)
randIndex(kfit$cluster, optdigits[,65], correct = TRUE)
    ARI
0.6812285

2. EM Clustering:

In this part, I used MATLAB and R to do the EM clustering, but both of these methods have encountered problems.

The MATLAB code is:

```
opttra = csvread('optdigits.tra');
opttes = csvread('optdigits.tes');

train = opttra(:,1:64);
test = opttes(:,1:64);
cl = opttra(:,65);

k = 10;
Sigma = {'diagonal','full'};
nSigma = numel(Sigma);
SharedCovariance = {true,false};
SCtext = {'true','false'};
nSC = numel(SharedCovariance);

options = statset('MaxIter',1000); % Increase number of EM iterations

for i = 1:nSigma;
    for j = 1:nSC;
        gmfit =
fitgmdist(train,k,'CovarianceType',Sigma{i},'SharedCovariance',SharedCovarian
ce{j},'Options',options);
        clusterX = cluster(gmfit,train);
        [aic,bic] = aicbic(clusterX,length(train),length(train));
        aic(1,:)
        bic(1,:)
    end
end
```

When do the em clustering, I used the with shared and not shared covariance matrices among components, and with diagonal and non-diagonal ("full") covariance matrices.
However, the em clustering can't execute correctly, I have got the error message as following:
Error using gmdistribution.fit (line 40)
The following column(s) of data are effectively constant: 1 40.

Error in fitgmdist (line 121)
gm = gmdistribution.fit(X,k,varargin{:});

When I looked at the column 1 and 40 of the training dataset, I have found that both of these columns are **zeros**. The gmdistribution.fit can not use those kind data to compute.

In R part:
I have used two different libraries to do the EM clustering in R, however, both of the methods can't do the process according to the question requirements.

```
library(EMCluster)
ret.em <- init.EM(mydata, nclass = 10, method = "em.EM")
ret.Rnd <- init.EM(mydata, nclass = 10, method = "Rnd.EM", EMC = .EMC.Rnd)
emobj <- simple.init(mydata, nclass = 10)
ret.init <- emcluster(mydata, emobj, assign.class = TRUE)
par(mfrow = c(2, 2))
plotem(ret.em, x)
plotem(ret.Rnd, x)
plotem(ret.init, x)
```

During using EMCluster, when I execute the init.EM function, then the computer was halt and has no reaction. And if I using simple.init, and the result has a warning message:

```
Warning message:
In emcluster(mydata, emobj, assign.class = TRUE) :
```

A stable solution is not avaliable.

```
library(mclust)
msEst <- mstep(modelName = "VVV", data = optdigits[,-65],z = unmap(optdigits[,65]))
emmodel <- estep(modelName = msEst$modelName, data = optdigits[,-65], parameters =
msEst$parameters)
```

When using library mclust, I also have a warning message:

```
attr(,"WARNING")
[1] "cannot compute E-step"
attr(,"returnCode")
[1] -1
```

```
# Another method using Mcluster only has one step
emmodel <- Mclust(optdigits[,1:64], G = 2, modelNames = 'EII')
# clustering results:
emmodel$classification
```

Section B: Nonparametric Methods

1. Univariate Density Estimation:

Randomly generate a set of N=100 data points using a uniform distribution in the range from 0 to 50.

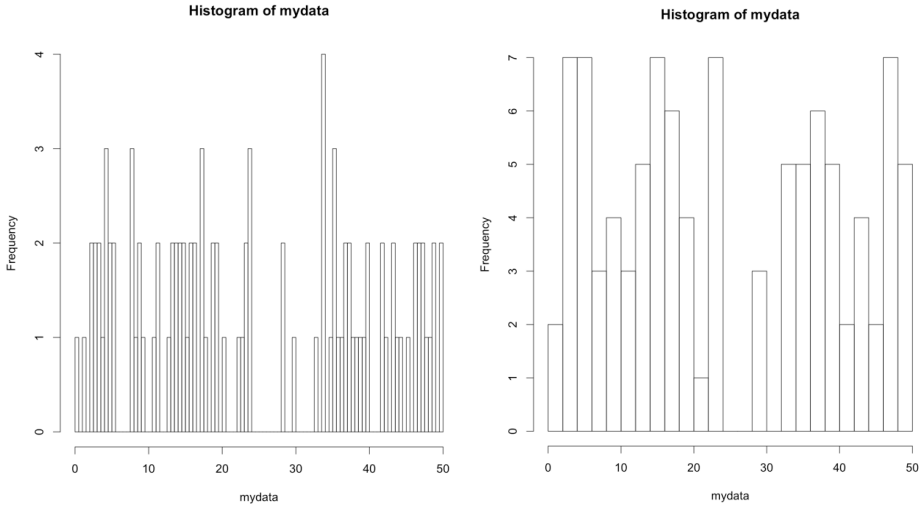
mydata <- runif(100, min=0, max=50)

(1) Naïve Estimator:

bin width h= 1 and bin width h=4:

hist(mydata,100)

hist(mydata,25)

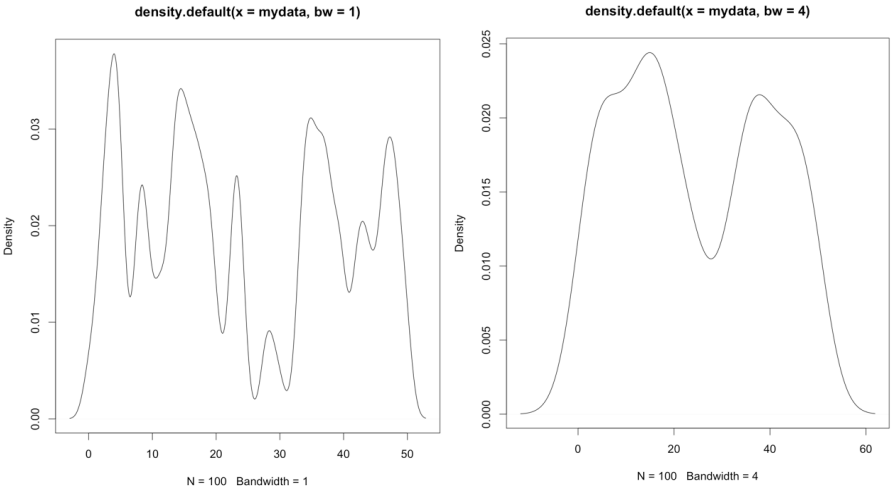


(2) Gaussian kernel estimator

bin width h= 1 and a separate plot with h=4

plot(density(mydata, bw = 1))

plot(density(mydata, bw = 4))

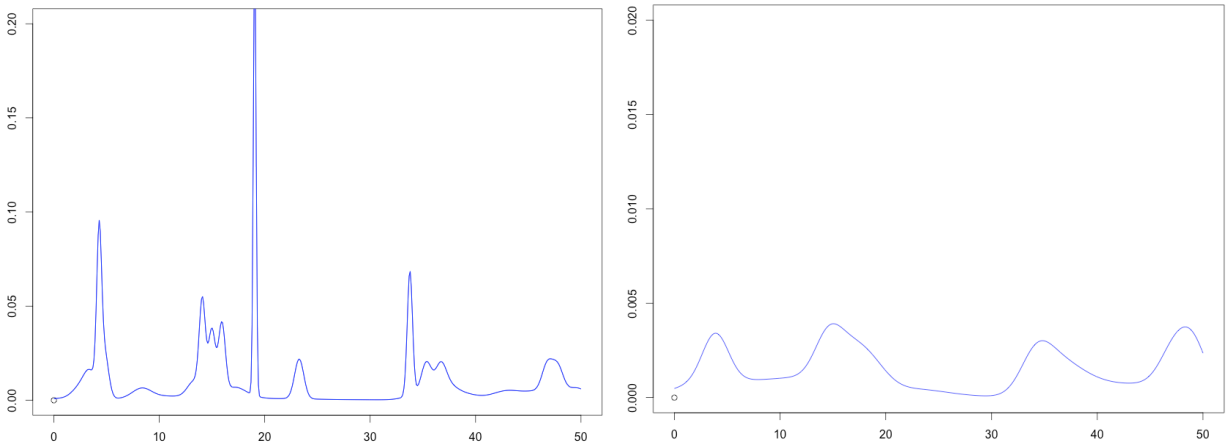


(3) k-nearest neighbor kernel estimator

I can't find k-nearest neighbor kernel estimator in R, then I coded it by myself:

```
naive_estimate <- function(data, k=3){
  xt <- numeric(ceiling(length(data)/k))
  dx <- numeric(ceiling(length(data)/k))
  data <- sort(data)
  # create the xt and dx vector
  for(i in 1:length(xt)){
    if(i == length(xt)){
      ind <- (i-1)*k + 1
      xt[i] <- mean(data[ind:length(data)])
      last = data[length(data)] - data[ind]
      if(last == 0){
        dx[i] <- 1
      }
      else{
        dx[i] <- last
      }
    }else{
      start <- (i-1)*k+1
      end <- i*k
      xt[i] <- mean(data[start:end])
      dx[i] <- data[end] - data[start]
    }
  }
  # Now calculate the kernels
  x <- seq(0, 50, 0.1)
  x_total <- numeric(length(x))
  for(i in 1:length(x_total)){
    for(j in 1:length(xt)){
      x_total[i] <- x_total[i] + dnorm(x[i],mean = xt[j], sd = dx[j])/(length(data)*dx[j])
    }
  }
  lines(x, x_total, col=4)
}

plot(0, 0, xlim=c(0, 50), ylim=c(0, 0.2), ylab="", xlab="K=3")
naive_estimate(mydata, k = 3)
plot(0, 0, xlim=c(0, 50), ylim=c(0, 0.02), ylab="", xlab="K=6")
naive_estimate(mydata, k = 6)
```



II Nonparametric Classification:

I can't find KNN using cosine distance, then I have wrote a cosine distance by myself:

```
library(lsa)

Cosine_KNN<-function(train,test, cl, k=1) {
  dist <- apply(train,1,function(trarow){
    apply(test,1,function(tesrow){
      cosine(as.numeric(trarow), as.numeric(tesrow))
    })
  })

  k_nearest<-apply(dist,2, function(x) {
    k_n<-which(x %in% sort(x)[1:k]);
    classes<-as.numeric(names(which.max(table(cl[k_n]))))
    return (factor(classes,levels = 0:9))
  })
}

knn <- Cosine_KNN(train, test, optdigits_tra[,65], k=1)
cl <- optdigits_tra[,65]
confusionMatrix(cl, knn)
```

Model	K	Confusion Matrix (Accuracy)
Cosine	1	0.079
	5	0.051
	9	0.0662
	11	0.0759
Euclidean	1	0.98
	5	0.9783
	9	0.9777
	11	0.9783

Mahalanobis	1	
	5	
	9	
	11	

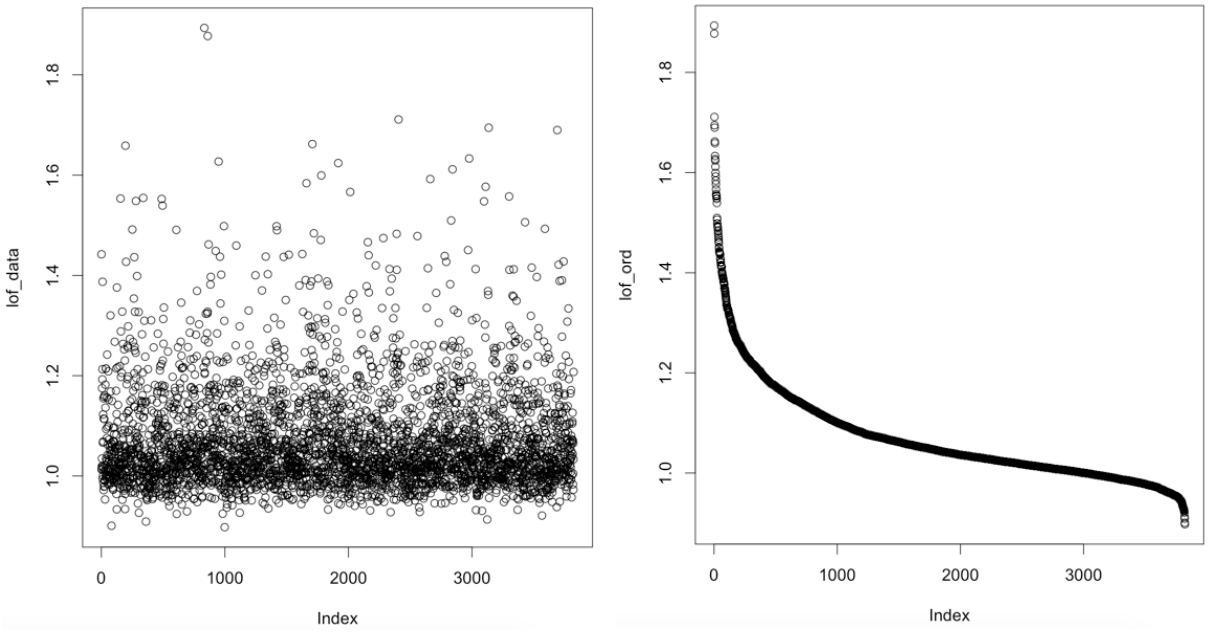
```
library(knnGarden)
knnMCN(TrnX = train, OrigTrnG = cl, TstX = test,K = 5)
Error in knnMCN(TrnX = train, OrigTrnG = cl, TstX = test, K = 5) :
Warnings:
sample variance-covariance matrix is singular,
and larger class sample capacity is required ,
or you can try other methods in knnWXM of this package
Can't find k-nearest neighbour classification of Mahalanobis Distance version
```

From the result above, we can see that using knn with Euclidean distance has very good accuracy when do the confusion matrix.

III Outlier Detection:

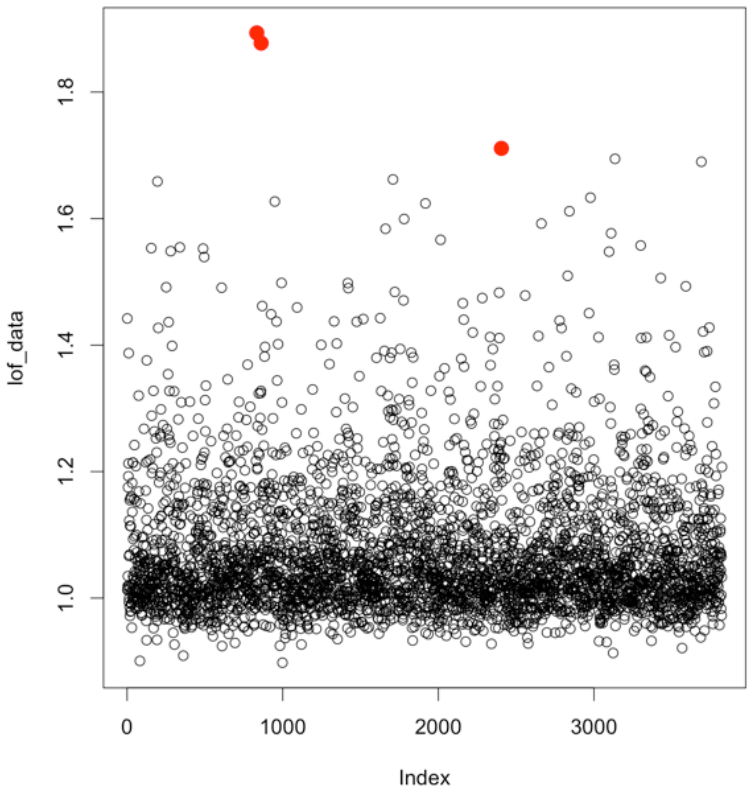
The local outlier factor (LOF) is a measure of outlierness that is calculated for each observation. The user decides whether or not an observation will be considered an outlier based on this measure. The LOF takes into consideration the density of the neighborhood around the observation to determine its outlierness. In this section, I will use Rlof package to do the Local Outlier Factor analysis.

```
library(Rlof)
lof_data <- lof(train, k=3)
lof_ord <- order(lof_data,decreasing = T)
plot(lof_data[lof_ord])
```



The left above graph is an unsorted graph, and the right above is a sorted graph. We can see that there has an obvious “elbow” in the sorted graph.

Take the 3 data instances with the highest LOF values:
points(x = lof_ord[1:3],y = lof_data[lof_ord][1:3], col='red', cex=1.5, pch=19)



IV Nonparametric Regression:

lowesss = smooth(opttra(1:1911,:),opttra(1912:3822,:), 0.2);
plot(lowesss)