HW9

Guannan Shen

November 29, 2018

Contents


```
## set up workspace
library(class)
library(knitr)
library(tidyverse)
library(magrittr)
library(stats)
options(stringsAsFactors = F)
options(dplyr.width = Inf)
getwd()
```

```
## [1] "/home/guanshim/Documents/Stats/CIDA_OMICs/7659Stats_Genetics/HW9"
```

```
## not in function
"%nin%" <- Negate("%in%")
```

```
# ######## clean memory ###################### rm(list =
# ls()) gc() slotNames(x) getSlots(x)
```

1 Classification

• Download the data provided on Canvas (dataHW9-breastcancer.Rdata). This file contains the "breastcancer" object, which is a dataset for 46 breast tumor samples where 23 are positive for an estrogen receptor (ER+) and 23 were negative (ER-) (West et al., PNAS 2001 98:11462-11467). There are expression levels for 7129 genes for each sample in this list (x) and class labels for each sample (y).

• Install the class package from CRAN

1.1 (a) Describe the k-nearest neighbor algorithm and how it classifies observations. Using the function knn() from the class package, run k-nearest neighbors with k = 3 to train and test on the same training data set (train). What percentage of subjects were correctly classified? It is not good practice to train and test on the same data set, why not?

The KNN here is a non-parametric method used for classification. The output is a class label. An object is classified by a majority vote of its k nearest neighbors, and the classes of neighbors are known. Thus, this is a type of supervised learning.

This method using training examples with features and class labels. When a test point (have features but does not have a class label) is provided, the distance metric (such as Euclidean distance, correlation coefficients) is calculated to define its k nearest neighbors. Eventually, the test point is assigned to the majority labels of its neighbors.

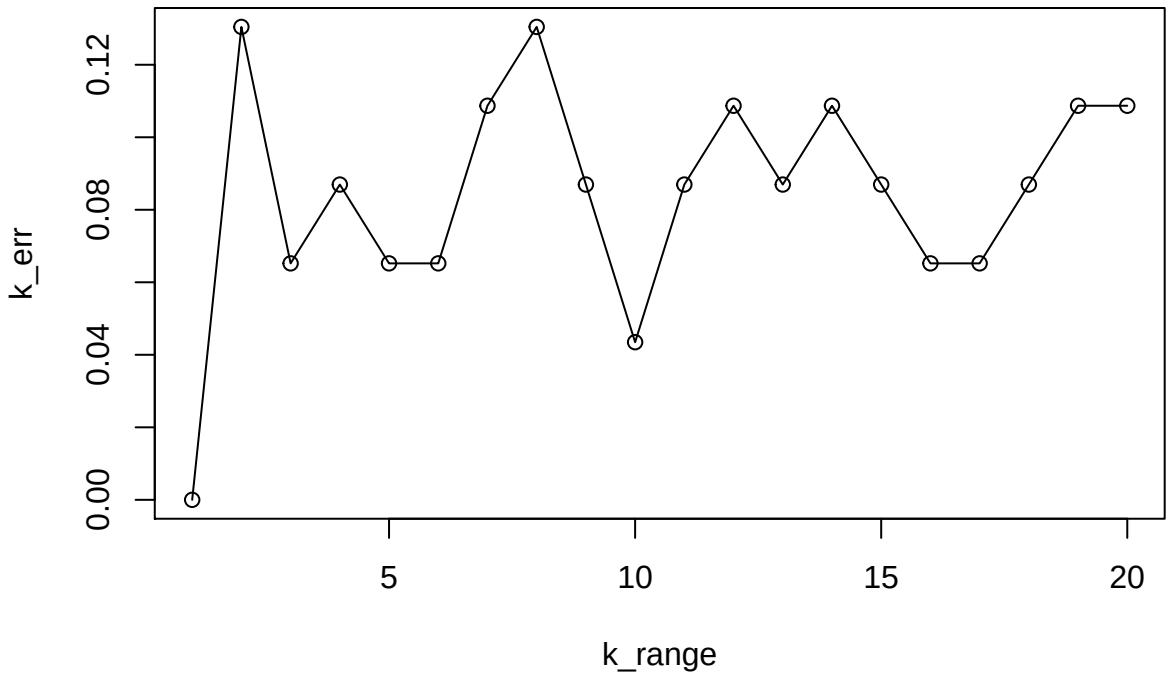
```
load("dataHW9_breastcancer.Rdata")
## import train test data
train <- breastcancer[[1]]  #training expression data
trainclass <- breastcancer[[2]]  #training classes
test <- newpatients  #new expression data
testclass <- trueclasses  #new classes

## run knn()
k1 <- knn(t(train), t(train), trainclass, k = 3)
k1_err <- sum(!(k1 == trainclass))/length(k1)
k1_err
```

```
## [1] 0.06521739
```

```
## k selection k is usually small
set.seed(123)
k_err <- NULL
k_range <- 1:20
for (i in k_range) {
    knn = knn(t(train), t(train), trainclass, k = i)
    k_err[i] <- sum(!(knn == trainclass))/length(knn)
}

## double check
sum(!(knn(t(train), t(train), trainclass, k = 1) == trainclass))
```

```
## [1] 0
```

```
## plot
plot(k_range, k_err, type = "o")
```




```
### test with new data
k_err_test <- NULL
for (i in c(1, 6, 10)) {
    knn = knn(t(train), t(test), trainclass, k = i)
    k_err_test[which(c(1, 6, 10) %in% i)] <- sum(!(knn == testclass))/length(knn)
}
k_err_test
```

```
## [1] 0.3333333 0.0000000 0.0000000
```

In this case, 93.4782609% of subjects were correctly classified.

If we train and test on the same data set, the result always looks too good, thus we can not test how well the model actually is. And we would never know if this model is generalizable or not. This approach cannot adjust for the overfitting issue.

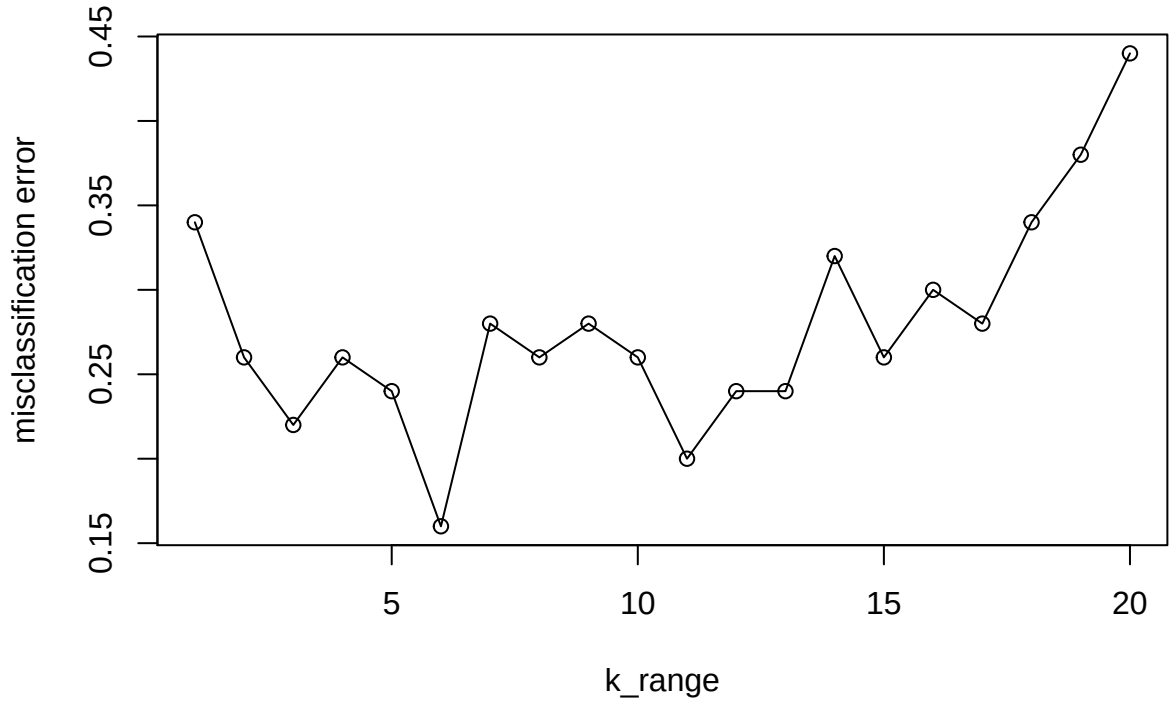
1.2 (b) Repeat part a) but for multiple values of k. What values and range of k are suitable for k-nearest neighbor? Plot k versus error rate. What value of k would you select based on this plot and why?

The k is usually a small number, but larger values of k reduces effect of the noise on the classification. Also, in binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes. Since I don't fully trust the train and test with the same dataset approach. In this case, based on the above plot, I would choose k = 10. Because using k = 1 in this approach will always have a 0 error rate as the data point are classified based on itself.

1.3 (c) Using k selected in part b), predict the tumor class for three new subjects in newpatients using their k nearest-neighbors in the training data. The correct classes are in trueclasses. How well did you do?

In my case, k=10 has an error rate 0.

```
## 5 fold cross validation
set.seed(123)
k_err <- NULL
k_err_fold <- NULL
k_range <- 1:20
for (k in k_range) {
    for (j in 1:5) {
        if (j <= 4) {
            knn = knn(t(train)[-(((j - 1) * 10 + 1):(10 * j)),
                ], t(train)[((j - 1) * 10 + 1):(10 * j), ], trainclass[-(((j -
                1) * 10 + 1):(10 * j))], k = k)
            k_err_fold[j] <- sum(!(knn == trainclass[((j - 1) *
                10 + 1):(10 * j)]))/length(knn)
        } else {
            knn = knn(t(train)[1:36, ], t(train)[37:46, ], trainclass[1:36],
                k = k)
            k_err_fold[j] <- sum(!(knn == trainclass[37:46]))/length(knn)
        }
    }
    k_err[k] <- mean(k_err_fold)
}


## plot
plot(k_range, k_err, type = "o", ylab = "misclassification error")
```



```
## true test see above
```

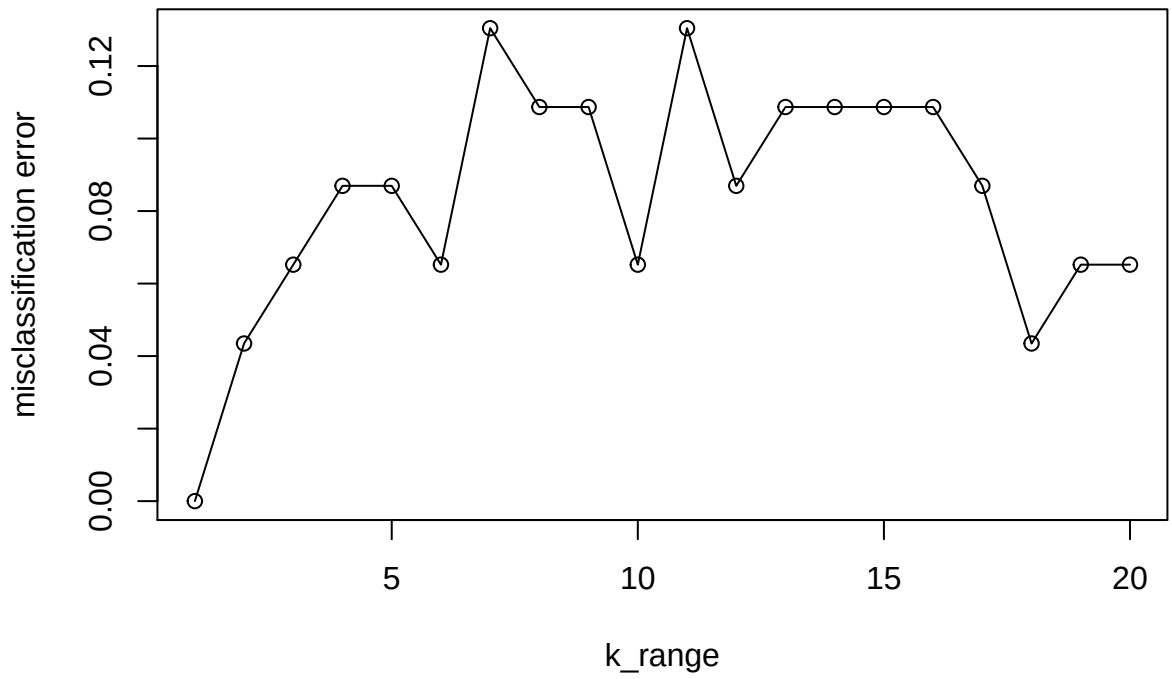
1.4 (d) Extra credit: Perform 5-fold cross validation to determine your error rate on the training data (perform on only one random partitioning). Since 46/5 is not an integer, make sure that the test group is the one with 10 patient samples. Plot your results for different values of k. How does this compare to the error rates from part a) above.

With 5-fold cross validation by taking the mean error rates of 5 folds, the k = 6 was my choice. In this case, the overall error rates were higher compared with previous approach, because we used a pseudo test dataset. Once the scenario was more like the real training and testing situation, we were able to find more reliable value of the tuning parameter k.

1.5 (e) Extra Credit: Write your own code for k-nearest neighbor classification using one minus the absolute correlation as a distance. Apply your code to the samples and plot the error rate for different values of k. What value for k do you select based on the training data? How does that compare with the results using the Euclidean distance from part a) and b).

```
knn_corr <- function(train, test, label, k) {
    train = as.matrix(train)
    test = as.matrix(test)
    k = as.integer(k)
    ntrain = nrow(train)
    ntest = nrow(test)
    if (ntrain == length(label)) {
        disCorr = matrix(NA, nrow = ntrain, ncol = ntest)
        for (i in 1:ntest) {
            for (j in 1:ntrain) {
                disCorr[j, i] = 1 - cor(train[j, ], test[i, ],
                  method = "pearson")
            }
        }
        disCorr = as.data.frame(disCorr)
        disCorr[, (ntest + 1)] = label
        knnlabel = NULL
        for (i in 1:ntest) {
            topk = disCorr[order(disCorr[, i], decreasing = F)[1:k],
                c(i, (ntest + 1))]
            knnlabel[i] = names(which.max(table(topk[, 2])))
        }
        knnlabel
    } else {
        return("Check the length of label!")
    }

}

## test new knn
set.seed(123)
```

```r
k_err <- NULL
k_range <- 1:20
for (i in k_range) {
    knn = knn_corr(t(train), t(train), trainclass, k = i)
    k_err[i] <- sum(!(knn == trainclass))/length(knn)
}

## plot
plot(k_range, k_err, type = "o", ylab = "misclassification error")
```



Using one minus the absolute correlation as the distance and the same approach in (a) and (b), k = 2 seems to be a reasonable choice and the results were quite different from the previous method.

2 Clustering

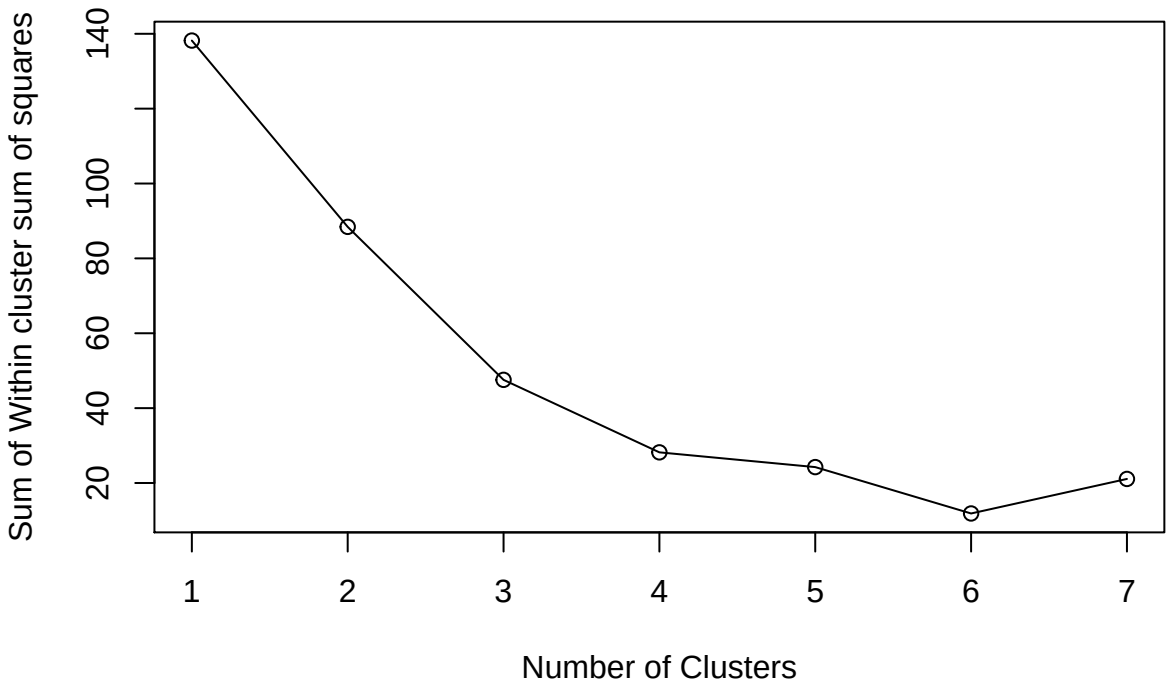
Download the data provided on Canvas (dataHW9-cellcycle.txt) for yeast gene expression over two cell cycles (Cho et al., Molecular Cell 1998, 2:65-73). In this experiment, mRNA was extracted from yeast cells at 10 minute intervals after reinitiation of the cell cycle. The data entries are fluorescence intensities for a single dye at each time point. The first column is the yeast gene ID and the second column is the gene name. HSP genes encode heat shock proteins, RPS genes encode ribosomal proteins and MCM genes encode for members of the MCM (mini-chromosome maintenance) complex.

Read sections 10.4 and 11.5.1 provided on Canvas (Computational Genome Analysis, Deonier, Tavare & Waterman, 2005).

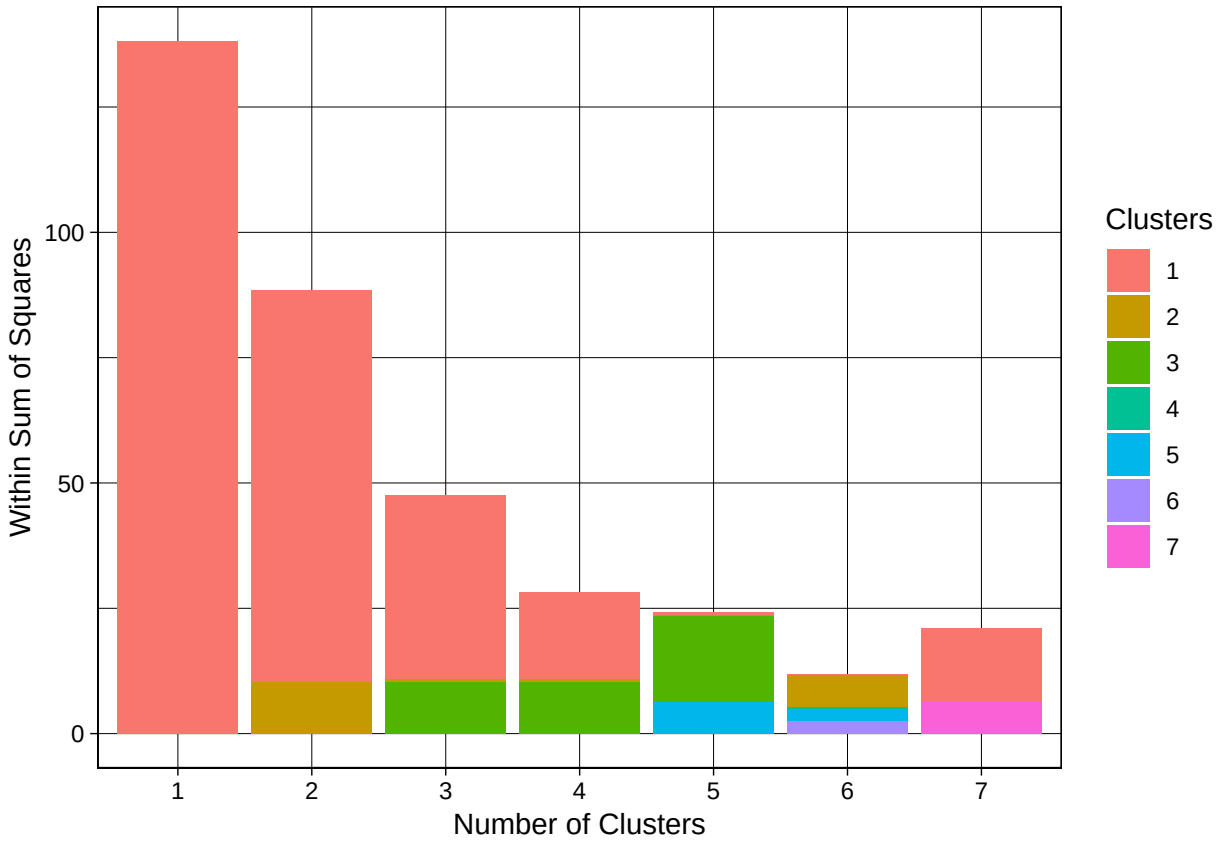
The stats package contains both the hclust() and kmeans() functions.

```r
# load txt data
yeast_cellcycle <- read.delim("dataHW9-cellcycle.txt", header = F,
    sep = "\t")
yeast <- yeast_cellcycle[, 3:18]
rownames(yeast) <- yeast_cellcycle[, 2]
```

```
########### k means method ############ standardized data for k means
syeast <- apply(yeast, 1, function(x) {
    (x - mean(x))/sd(x)
})

## kmeans and try k
maxk <- 7
k_c <- 1:maxk
k_sws <- NULL

k_wsm <- NULL
for (i in k_c) {
    km <- kmeans(t(syeast), i, iter.max = 10)
    k_sws[i] <- sum(km$withinss)
    if (i == 1) {
        k_wsm <- c(km$withinss)
    } else {
        k_wsm <- c(k_wsm, km$withinss)
    }
}
plot(k_c, k_sws, type = "o", xlab = "Number of Clusters", ylab = "Sum of Within cluster sum of squares")
```




```
## using all wsm stacking bar plot
k_sall <- data.frame(matrix(NA, nrow = sum(1:maxk), ncol = 3))
k_sall$X1 <- as.factor(rep(1:maxk, 1:maxk))
k_sall$X2 <- sapply(1:maxk, function(x) seq(1:x)) %>% unlist() %>%
    as.factor()
k_sall$X3 <- k_wsm

ggplot(data = k_sall, aes(x = X1, y = X3, fill = X2)) + geom_bar(stat = "identity") +
    labs(fill = "Clusters") + labs(x = "Number of Clusters") +
    labs(y = "Within Sum of Squares") + theme_linedraw()
```



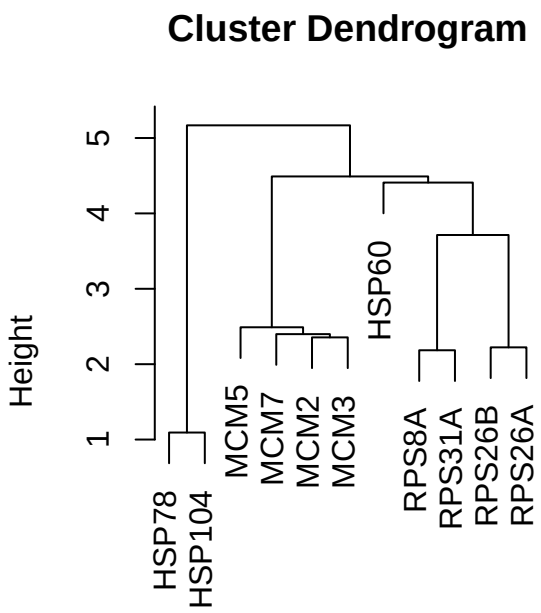

```
## choice of 4
km4 <- kmeans(t(syeast), 4, iter.max = 10)
km4$withinss
```

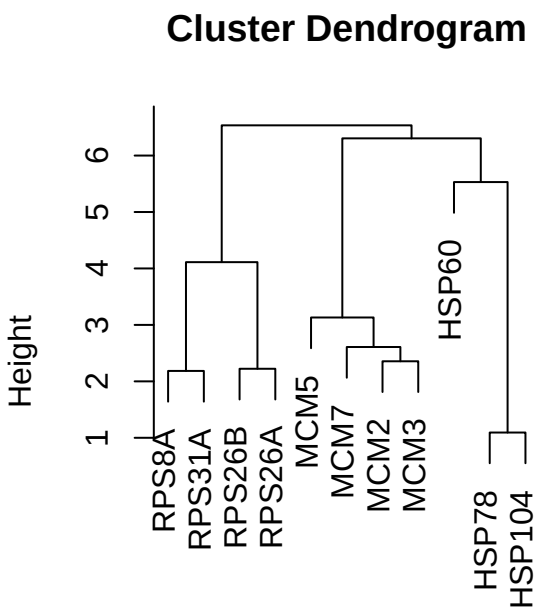
```
## [1] 10.3438577 14.6732950  0.5977036  2.4705677
```

```
km4$cluster
```

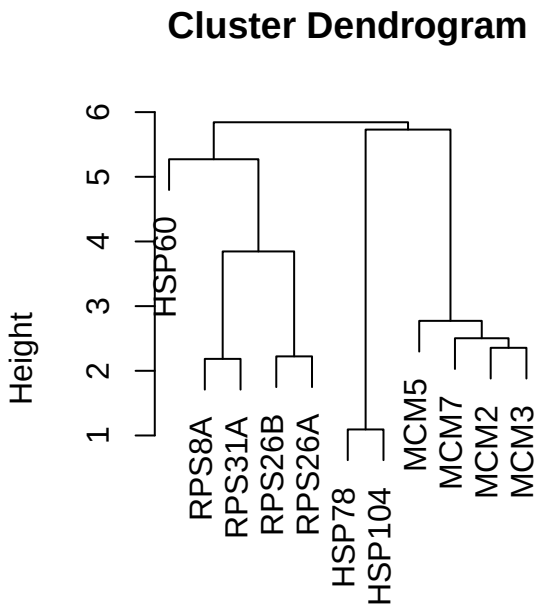
```
##   MCM2  RPS8A   MCM7  HSP78   MCM3 RPS26B RPS26A RPS31A HSP104  HSP60
##      1      2      1      3      1      4      4      2      3      2
##   MCM5
##      1
```

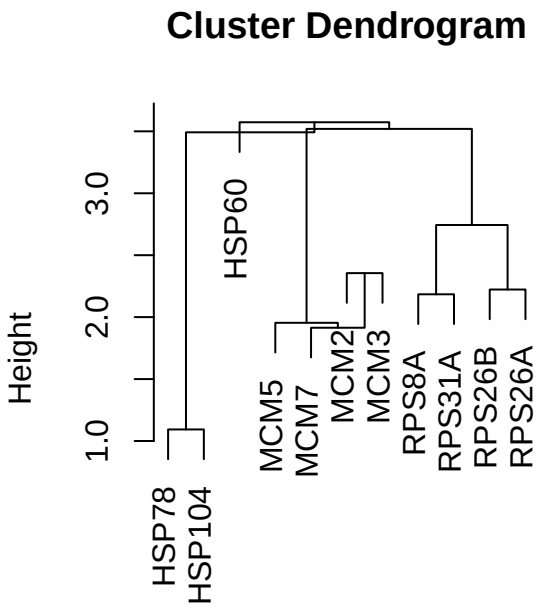
```
################ hierarchical clustering ############ get distance matrix
m <- as.matrix(dist(t(syeast), method = "euclidean"))
d <- as.dist(m)
# hierarchical clustering
par(mfrow = c(1, 2))
plot(hclust(d, "single"), xlab = "Yeast", main = "Cluster Dendrogram")
plot(hclust(d, "complete"), xlab = "Yeast", main = "Cluster Dendrogram")
```



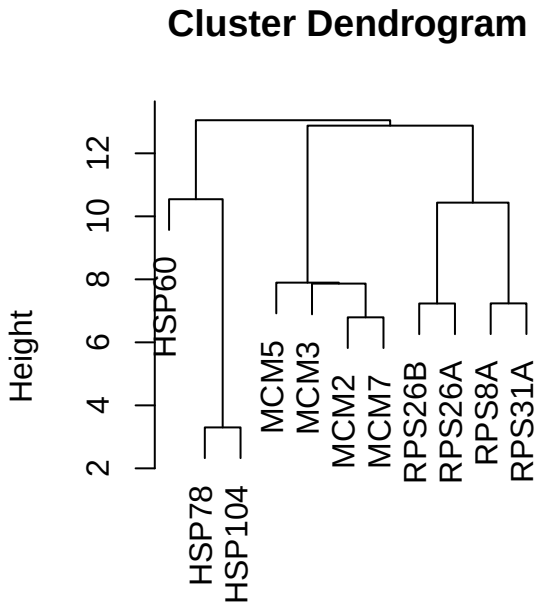


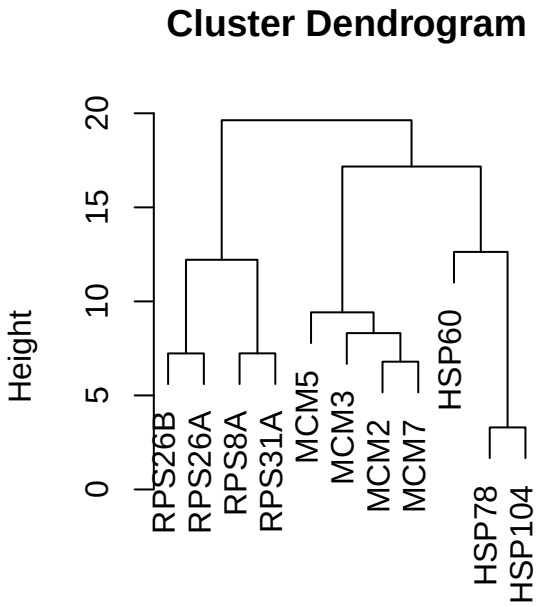

```
plot(hclust(d, "average"), xlab = "Yeast", main = "Cluster Dendrogram")
plot(hclust(d, "centroid"), xlab = "Yeast", main = "Cluster Dendrogram")
```



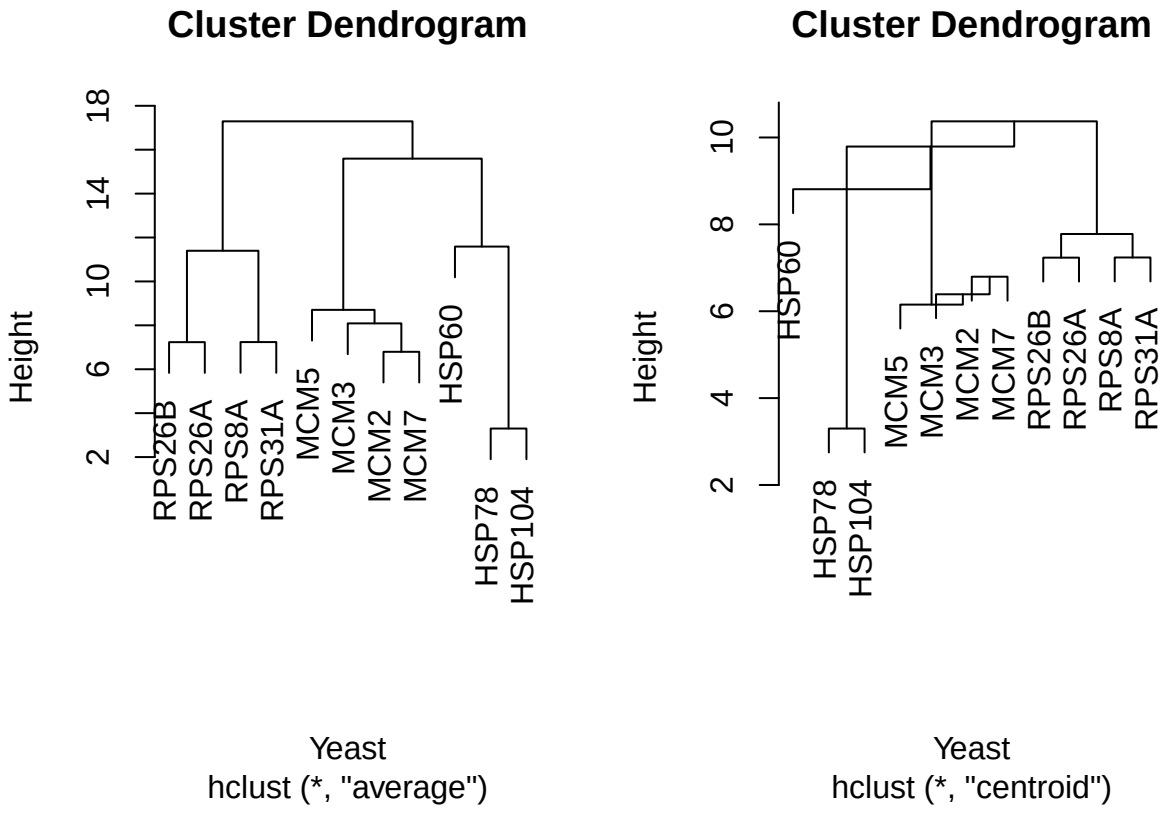


```
## manhattan distance
m <- as.matrix(dist(t(syeast), method = "manhattan"))
d <- as.dist(m)
# hierarchical clustering
plot(hclust(d, "single"), xlab = "Yeast", main = "Cluster Dendrogram")
plot(hclust(d, "complete"), xlab = "Yeast", main = "Cluster Dendrogram")
```






```
plot(hclust(d, "average"), xlab = "Yeast", main = "Cluster Dendrogram")
plot(hclust(d, "centroid"), xlab = "Yeast", main = "Cluster Dendrogram")
```



2.1 (a) Using the R code in section 11.5.1 as an example, run the k-means algorithm to cluster the data (use the standardized data - see Step 2). Try different values for k = 2, 3 ... and create a plot of "Within Sum of Squares" versus k. What is your best choice of k? For the final clusters, what genes are grouped together?

Based on above plots on sum of "Within Sum of Squares" and stacking barplot of "Within Sum of Squares", k = 4 is my best choice of k. In this 4-cluster clustering, I got 4 MCMs in the 4th cluster, HSP78 and HSP104 in the 3rd cluster, RPS26B and RPS26A in the 2nd cluster and RPS8A, RPS31A and HSP60 in the 1st cluster.

2.2 (b) Using the R code in section 10.4 as an example, apply the hierarchical clustering algorithm on the data. Calculate the Euclidean distance (using dist() then as.dist()) with standardized data. How does the cluster membership obtained with hierarchical clustering compare with the results from k-means in part a)? Describe the "method" option in hclust discussed in class. Try different values for "method". What happens when you change this option? Repeat this analysis with the Manhattan (L1) distance instead. How do the results change?

By using the example method of hierarchical clustering, "single", this result was very similar with the grouping obtained by k=4, kmeans method. In detail, all 4 MCMs were grouped together, HSP78 and HSP104 were in a cluster, RPS26B and RPS26A were in another one, and RPS31A and RPS8A were in one cluster. The

difference here is HSP60 was in its own cluster. However, similar to the kmeans clustering, this one was close to the cluster of RPS31A and RPS8A, not the cluster of HSPs.

In the class, we talked about 4 methods to compare two clusters.
1. Single linkage using minimum distance between points in different clusters.
2. Complete linkage using maximum distance.
3. Mean linkage using average of all distances.
4. Centroids using distance between within-cluster means.

Through different methods, the key difference was how was the unique one, HSP60 clustered. Other clusters were very robust. Overall, the "complete" method had the best clustering based on the labels, where the HSP60 was grouped into the HSP cluster.

In summary, the manhattan (L1) distance out-performed the euclidean distance methods. Most hierarchical clustering methods based on the manhattan distance can correctly group HSP60 into the HSP cluster.