Unsupervised learning - clustering and dimension reduction

Anna Yeaton

Fall 2018

Lab Section

Download auto data from the github page https://github.com/ayeaton/BMSC-GA-4439-Fall2018/blob/master/Auto.data.txt or the *Statistical Learning* book website here: http://www-bcf.usc.edu/~gareth/ISL/data.html

Today, we are going over Hierarchical clustering, K-Means Clustering, PCA, ICA, and NMF.

```
#set your working directory
#setwd("")
Auto_data <- read.table("Auto.data.txt", header=T, stringsAsFactors = F)

View(Auto_data)

#remove cars with unknown horsepower and set horsepower to numeric
Auto_data <- Auto_data[-which(Auto_data$horsepower == "?"),]
Auto_data$horsepower <- as.numeric(Auto_data$horsepower)
str(Auto_data)
```

```
## 'data.frame':    392 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : chr  "chevrolet chevelle malibu" "buick skylark 320" "plymouth satellite" "
```

```
str(Auto_data$horsepower)
```

```
##  num [1:392] 130 165 150 150 140 198 220 215 225 190 ...
```

```
#save car names
Auto_data_names <- Auto_data$name

#use only fields with numeric values
Auto_data_clust <- Auto_data[,1:8]

dim(Auto_data_clust)
```

```
## [1] 392   8
```

```
print("Auto_data_clust")
```

```
## [1] "Auto_data_clust"
```

```
str(Auto_data_clust)
```

```
## 'data.frame':    392 obs. of  8 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
#392 is too much for a demo, so lets take the first 25
Auto_data_clust <- Auto_data_clust[1:25,]
rownames(Auto_data_clust) <- Auto_data_names[1:25]
print("ONLY the first 25 Auto_data_clust")
```

```
## [1] "ONLY the first 25 Auto_data_clust"
```

```
str(Auto_data_clust)
```

```
## 'data.frame':    25 obs. of  8 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
```

Hierarchical agglomerative clustering

Step 1. Assign each item to it's own cluster. We start with 25 clusters, one for each car.

Step 2. Calculate a similarity matrix between each cluster.

Step 3. Find the pair of clusters closest in similarity.

Step 4. Merge these clusters/recalculate similarity between clusters. Options are: single linkage (nearest neighbor), complete linkage (furthest neighbor), average linkage (mean distance between all pairs of data from the two different clusters), centroid linkage (distance between the means of all points in the clusters). Now we have 24 clusters.

Step 5. Repeat Step 3 and 4 until there is only one cluster.

In practice

Step 1. Each car is a cluster.

Step 2. Create a distance matrix from Auto_data_clust.

```
#help("dist")
hierarchical_dist <- as.matrix(dist(Auto_data_clust, method = "euclidean"))
#View(hierarchical_dist)
```

Step 3. Find the two cars that are the most similar to each other and print the names of those two cars

```
diag(hierarchical_dist) <- NA
arrayInd(which.min(hierarchical_dist), dim(hierarchical_dist))
```

```
##      [,1] [,2]
## [1,]   23   15
```

```
#postitions 23 and 15 are the most similar. Lets go back to the names of the cars
Auto_data_names[23]
```

```
## [1] "saab 99e"
```

```
Auto_data_names[15]
```

```
## [1] "toyota corona mark ii"
```

Step 4. Merge the two clusters together using average linkage.

```
#replace pos 15 with the average of pos 15 and 23
hierarchical_dist[,15] <- apply((hierarchical_dist[,c(23,15)]),1,mean)
hierarchical_dist[15,] <- apply((hierarchical_dist[c(23,15),]),2,mean)

#remove pos 23
hierarchical_dist <- hierarchical_dist[-23,-23]

#now position 15 represents the cluster containing the saab99e and the toyota corona mark ii
```

Step 5. To complete the algorithm, go back to step 3 and iterate through all of the previous steps until there are no more rows left

```
#Step 3. Find the two cars that are the most similar to each other and print the names of thos
diag(hierarchical_dist) <- NA
arrayInd(which.min(hierarchical_dist), dim(hierarchical_dist))
```

```
##      [,1] [,2]
## [1,]    4    3
```

```
#postitions 23 and 15 are the most similar. Lets go back to the names of the cars
Auto_data_names[4]
```

```
## [1] "amc rebel sst"
```

```
Auto_data_names[3]
```

```
## [1] "plymouth satellite"
```

```
#postitions 4 and 3 are the most similar

#replace pos 3 with the average of pos 3 and 4
hierarchical_dist[,3] <- apply((hierarchical_dist[,c(4,3)]),1,mean)
hierarchical_dist[3,] <- apply((hierarchical_dist[c(4,3),]),2,mean)

#remove pos 4
hierarchical_dist <- hierarchical_dist[-4,-4]

#now position 3 represents the cluster containing the amc rebel sst and the plymouth satellite
```

REPEAT. Find the two cars that are the most similar to each other and print the names of those two cars

```
diag(hierarchical_dist) <- NA
arrayInd(which.min(hierarchical_dist), dim(hierarchical_dist))
```

```
##      [,1] [,2]
## [1,]    4    3
```

```
#postitions 6 and 4 are the most similar. Lets go back to the names of the cars
Auto_data_names[6]
```

```
## [1] "ford galaxie 500"
```

```
Auto_data_names[4]
```

```
## [1] "amc rebel sst"
```

```
#postitions 6 and 4 are the most similar

#replace pos 4 with the average of pos 6 and 4
hierarchical_dist[,4] <- apply((hierarchical_dist[,c(6,4)]),1,mean)
hierarchical_dist[4,] <- apply((hierarchical_dist[c(6,4),]),2,mean)

#remove pos 4
hierarchical_dist <- hierarchical_dist[-6,-6]

#now position 6 represents the cluster containing the ford galaxie 500 and amc rebel sst
```

```
diag(hierarchical_dist) <- NA
arrayInd(which.min(hierarchical_dist), dim(hierarchical_dist))
```

```
##      [,1] [,2]
## [1,]    6    5
```

```
#postitions 5 and 4 are the most similar. Lets go back to the names of the cars
Auto_data_names[5]
```

```
## [1] "ford torino"
```

```
Auto_data_names[4]
```

```
## [1] "amc rebel sst"
```

```
#postitions 5 and 4 are the most similar

#replace pos 4 with the average of pos 5 and 4
hierarchical_dist[,4] <- apply((hierarchical_dist[,c(5,4)]),1,mean)
hierarchical_dist[4,] <- apply((hierarchical_dist[c(5,4),]),2,mean)

#remove pos 4
hierarchical_dist <- hierarchical_dist[-5,-5]

#now position 5 represents the cluster containing the ford torino and amc rebel sst
```

R function

Now that we know how the algorithm works, let's use the R function hclust. Plot the Dendogram resulting from clustering the Auto_data_clust using average linkage.

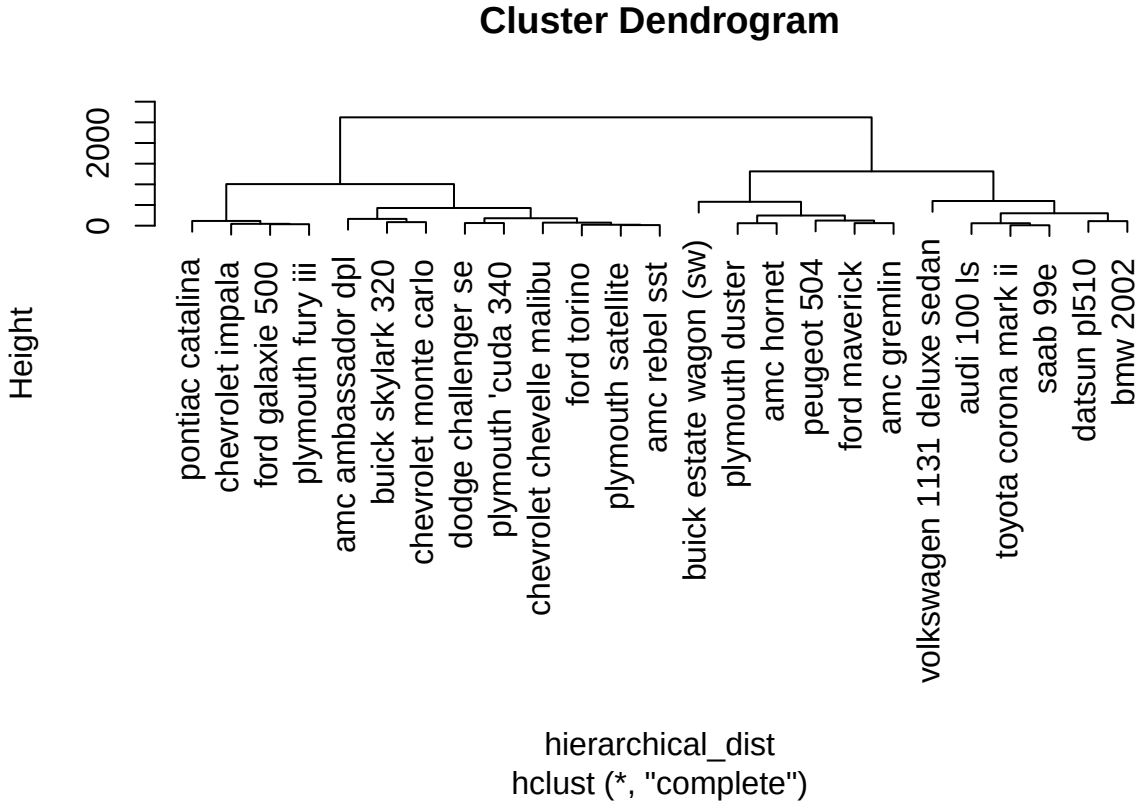
```
#help(hclust)
help(hclust)
hierarchical_dist <- dist(Auto_data_clust, method = "euclidean")
plot(hclust(hierarchical_dist), method = "average")
```

```
## Warning in graphics:::plotHclust(n1, merge, height, order(x$order), hang, :
## "method" is not a graphical parameter

## Warning in graphics:::plotHclust(n1, merge, height, order(x$order), hang, :
## "method" is not a graphical parameter

## Warning in axis(2, at = pretty(range(height)), ...): "method" is not a
## graphical parameter

## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...):
## "method" is not a graphical parameter
```


Cluster Dendrogram
Height
0
2000
pontiac catalina
chevrolet impala
ford galaxie 500
plymouth fury iii
amc ambassador dpl
buick skylark 320
chevrolet monte carlo
dodge challenger se
plymouth 'cuda 340
chevrolet chevelle malibu
ford torino
plymouth satellite
amc rebel sst
buick estate wagon (sw)
plymouth duster
amc hornet
peugeot 504
ford maverick
amc gremlin
volkswagen 1131 deluxe sedan
audi 100 ls
toyota corona mark ii
saab 99e
datsun pl510
bmw 2002
hierarchical_dist
hclust (*, "complete")

##K-Means Clustering

Step 1. Choose N number of clusters.

Step 2. Find the N items that are furthest apart and set them as cluster centroids.

Step 3. Assign one item in the dataset to the closest of the N cluster centroids.

Step 4. Recalculate the cluster centroid.

Step 5. Repeat Steps 3 and 4 until all items are in a cluster.

Step 6. Go through each item and reassess whether the item belongs in the current cluster or in a different cluster based on distance to cluster centroids. Every time an item is reassigned to a different cluster, the centroids must be recalculated.

Step 7. When every item belongs firmly to a cluster, or the iterations of Step 6 are endless, the algorithm is complete.

In practice

Step 1. We are going to cluster the 25 cars into two groups.

Step 2a. Find the two cars furthest from each other.

```
kmeans_dist <- as.matrix(dist(Auto_data_clust, method = "euclidean"))
diag(kmeans_dist) <- NA
arrayInd(which.max(kmeans_dist), dim(kmeans_dist))
```

```
##      [,1] [,2]
## [1,]   20    9
```

Step 2b. Create data frames to hold each cluster, cluster names, and centroids.

```
cluster_one <- Auto_data_clust[20,]
cluster_one_names <- Auto_data_names[20]
cluster_one_centroid <- cluster_one

cluster_two <- Auto_data_clust[9,]
cluster_two_names <- Auto_data_names[20]
cluster_two_centroid <- cluster_two
```

Step 3a. Sequentially put cars in either cluster one or cluster two. lets start with car 1. Is car 1 closer to cluster one or cluster two?

```
#distance to cluster 1
dist(rbind(cluster_one_centroid,Auto_data_clust[1,]), method = "euclidean")
```

```
##                            volkswagen 1131 deluxe sedan
## chevrolet chevelle malibu                     1684.301
```

```
#distance to cluster 2
dist(rbind(cluster_two_centroid,Auto_data_clust[1,]), method = "euclidean")
```

```
##                             pontiac catalina
## chevrolet chevelle malibu         937.6513
```

Step 3b. Add car 1 to cluster two and adjust the centroid value for cluster 2. The new centroid value is a mean of the values of cars in that cluster.

```
cluster_two <- rbind(cluster_two, Auto_data_clust[1,])
cluster_two_names <- rbind(cluster_two_names, Auto_data_names[1])
cluster_two_centroid <- apply(cluster_two,2,function(x) mean(as.numeric(x)))
```

Step 3d. Do for all cars

```
for(i in 1:nrow(Auto_data_clust)){
  if(i == 9 | i == 20){
    next
  }
  if(dist(rbind(cluster_two_centroid,Auto_data_clust[i,]), method = "euclidean") < dist(rbind(c
    cluster_two <- rbind(cluster_two, Auto_data_clust[i,])
    cluster_two_names <- rbind(cluster_two_names, Auto_data_names[i])
    cluster_two_centroid <- apply(cluster_two,2,function(x) mean(as.numeric(x)))
  }else{
    cluster_one <- rbind(cluster_one, Auto_data_clust[i,])
    cluster_one_names <- rbind(cluster_one_names, Auto_data_names[i])
    cluster_one_centroid <- apply(cluster_one,2,function(x) mean(as.numeric(x)))
  }
}
```

Step 4. Adjust the clusters by comparing the distance of each car to the centroid of its current cluster versus the distance to the centroid of the other cluster. Does it still belong in the current cluster? lets start with car 1. Does it belong in cluster two?

```
#does car 1 belong in cluster 2
dist(dist(rbind(cluster_one_centroid, Auto_data_clust[1,] )))
```

```
##                                1
## chevrolet chevelle malibu 1517.83
```

```
dist(dist(rbind(cluster_one_centroid, Auto_data_clust[1,] )))
```

```
##                                1
## chevrolet chevelle malibu 1517.83
```

Alright, so car 1 does belong in cluster two. To complete the algorithm, iterate over the cars until none of the cars switch clusters.

R function

Now we know how the algorithm works, lets use the R function kmeans.

```
help(kmeans)
kmeans_put <- kmeans(Auto_data_clust,2)
str(kmeans_put)
```

```
## List of 9
##  $ cluster     : Named int [1:25] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "names")= chr [1:25] "chevrolet chevelle malibu" "buick skylark 320" "plymoutl
##  $ centers     : num [1:2, 1:8] 15.4 22.8 8 5 374.8 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:8] "mpg" "cylinders" "displacement" "horsepower" ...
##  $ totss       : num 14590082
##  $ withinss    : num [1:2] 1883289 1407639
##  $ tot.withinss: num 3290927
##  $ betweenss   : num 11299155
##  $ size        : int [1:2] 13 12
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

Principal Components Analysis (PCA)

Principal Components Analysis is a linear dimensionality reduction algorithm. If you want to learn more about linear algebra, I suggest the MIT Open Courseware class here : https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/ There are two ways of doing PCA, Single Value Decomposition (SVD), and the method we will use today, using the covariance matrix of the data. SVD is not constrained to square matrices.

Step 1. Center data by subtracting the mean.

Step 2. Calculate covariance matrix of data.

Step 3. Perform Eigendecomposition of the covariance matrix. i.e. represent the matrix in terms of it's eigenvalues and eigen vectors

Step 4. Multiply the eigen vectors by the original data to express the data in terms of the eigen vectors.

Step 1. Center the data by subtracting the mean of the each column from the values in that column

```
#we need a square matrix
Auto_data_clust_pca <- data.matrix(Auto_data_clust[1:8,])

Center_auto <- apply(Auto_data_clust_pca, 1, function(x) x - mean(x))
str(Auto_data_clust_pca)
```

```
##  num [1:8, 1:8] 18 15 18 16 17 15 14 14 8 8 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:8] "chevrolet chevelle malibu" "buick skylark 320" "plymouth satellite" "amc
##   ..$ : chr [1:8] "mpg" "cylinders" "displacement" "horsepower" ...
```

Step 2. Calculate covariance matrix of the Auto data

```
Covariance_auto <- cov(Center_auto)
```

Step 3. Calculate eigen values and vectors

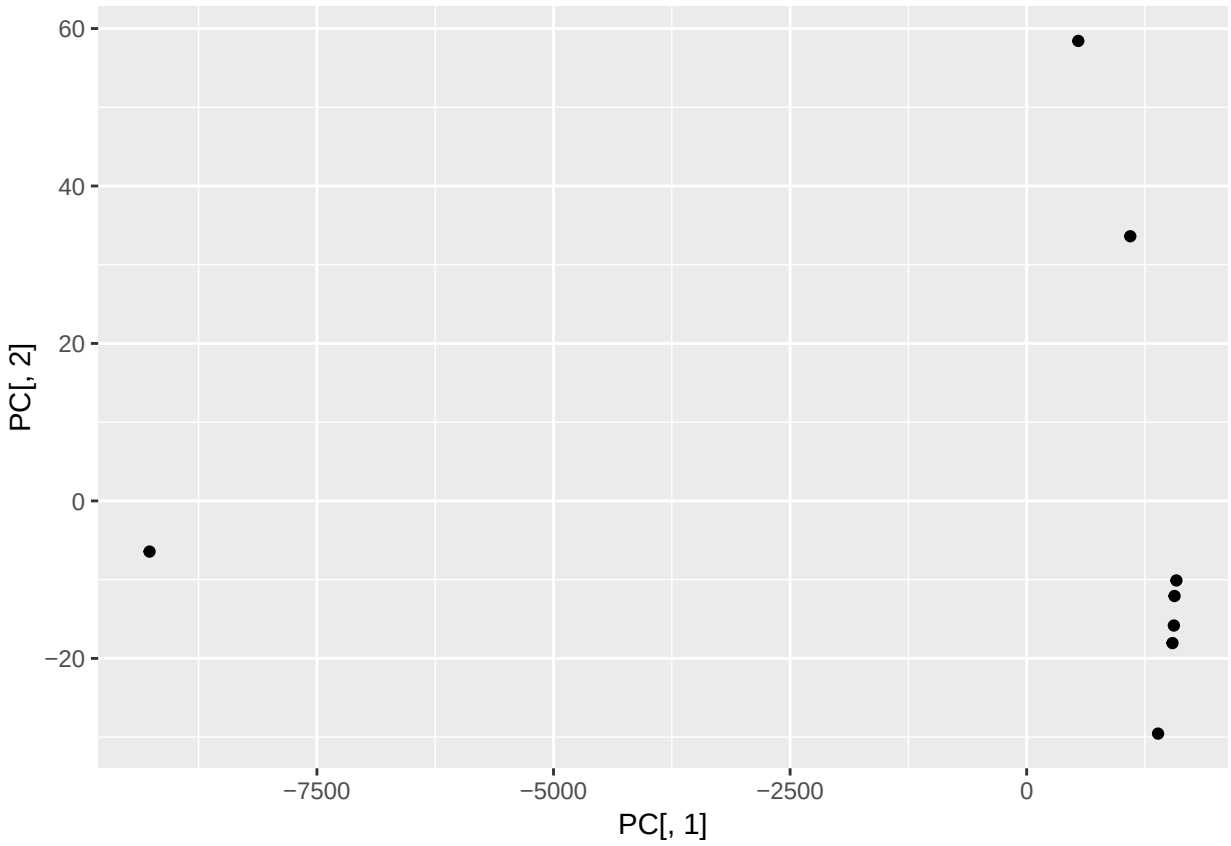
```
Eigen_value_auto <- eigen(Covariance_auto)$value

#columns are the eigen vectors
Eigen_vector_auto <- eigen(Covariance_auto)$vector
```

Step 4. Multiply the eigen vector matrix by the original data.

```
#?%*%
PC <- as.data.frame(data.matrix(Center_auto) %*% Eigen_vector_auto)

ggplot(PC, aes(PC[,1], PC[,2])) + geom_point(aes(PC[,1], PC[,2]))
```



Step 5. Find out which principal components explain the variance in the data.

```
#for each component, take the cumulative sum of eigen values up to that point and and divide by
round(cumsum(Eigen_value_auto)/sum(Eigen_value_auto) * 100, digits = 2)
```

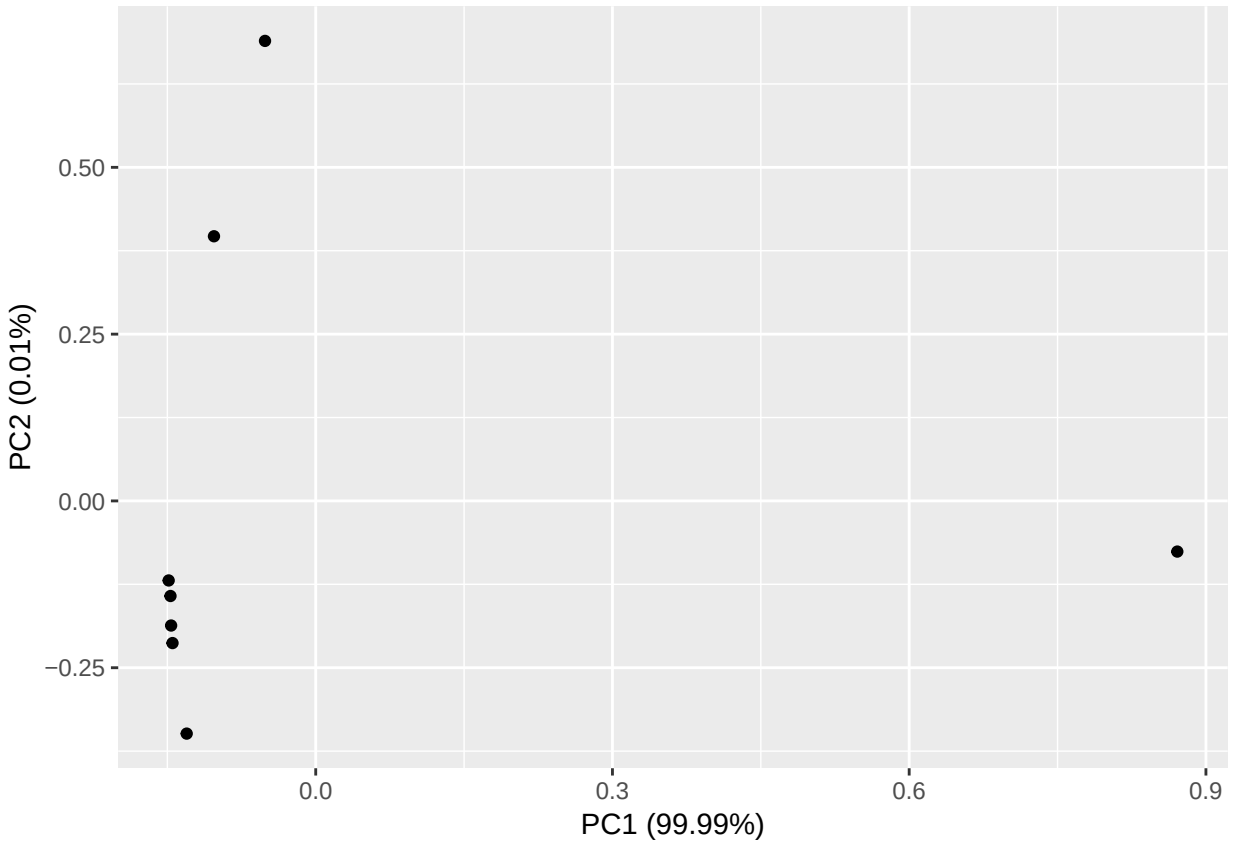
```
## [1]  99.99 100.00 100.00 100.00 100.00 100.00 100.00 100.00
```

Principal component 1 and 2 explain 99.99 percent of the variance. Principal component 1,2, and 3 together explain 100% of the variance in the data.

R function

Now that we know how PCA works, lets use the R funtion prcomp.

```
help("prcomp")
autoplot(prcomp(t(Auto_data_clust_pca)))
```


0.50
0.25
0.00
−0.25
PC2 (0.01%)
0.0
0.3
0.6
0.9
PC1 (99.99%)

Independent Component Analysis (ICA)

ICA is an algorithm that finds components that are independent, or subcomponents of the data.

Step 1. Whiten the data by projecting the data onto the eigen vectors (PCA).

Step 2. Solve the X=AS equation by maximizing non-gaussianty in the variables(components) in S.

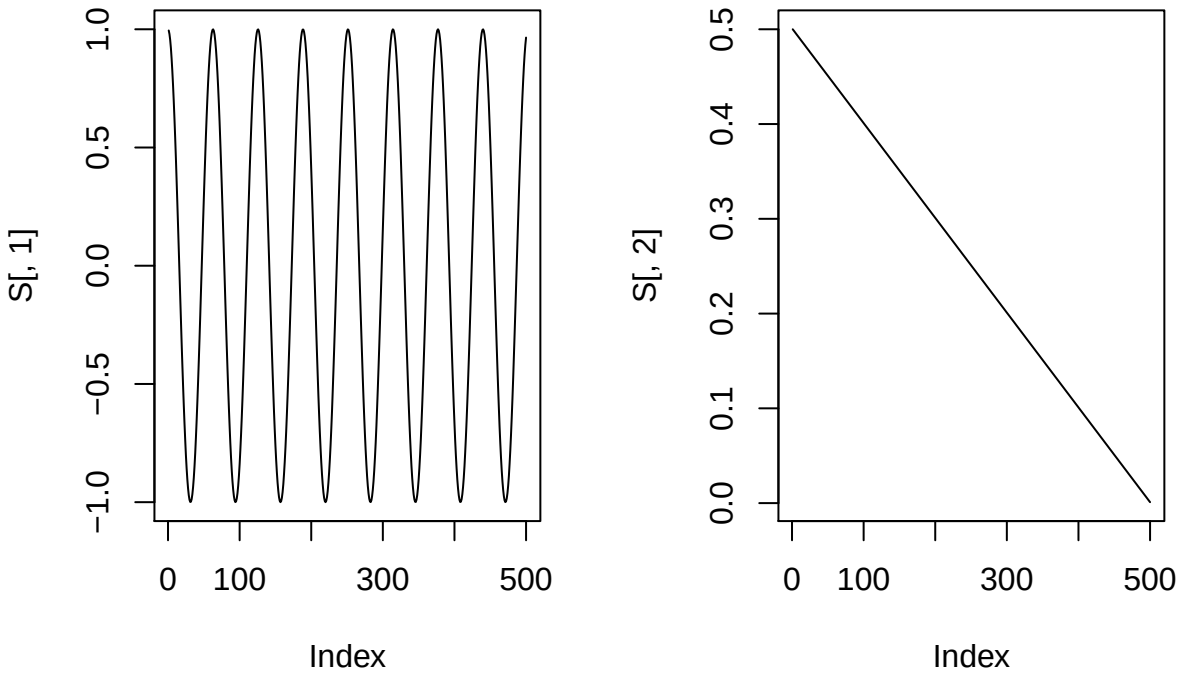
This results in a matrix S with components that are independent from each other.

We will use the fastICA algorithm.

But first we will go backwards. Create a matrix S with the independent components

```
#create two signals
S <- cbind(cos((1:500)/10), ((500:1)/1000))

par(mfcol = c(1, 2))
plot(S[,1], type="l")
plot(S[,2], type="l")
```

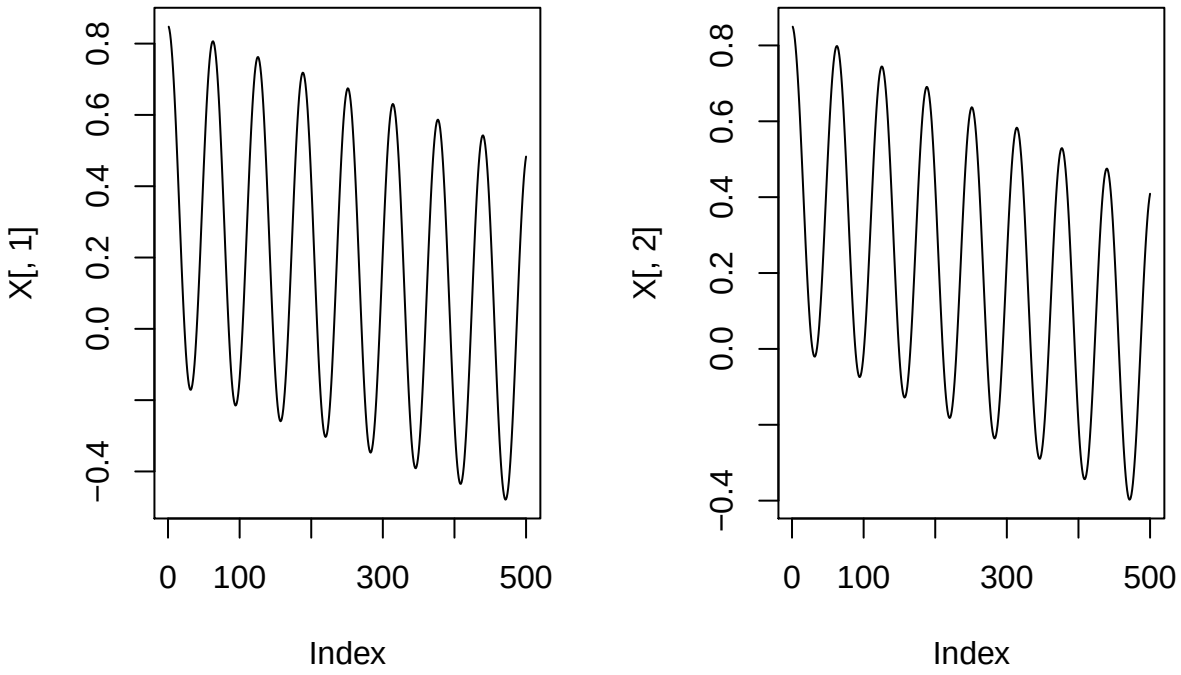


Create a mixing matrix A

```
A <- matrix(c(0.5, 0.7, 0.423, 0.857), 2, 2)
```

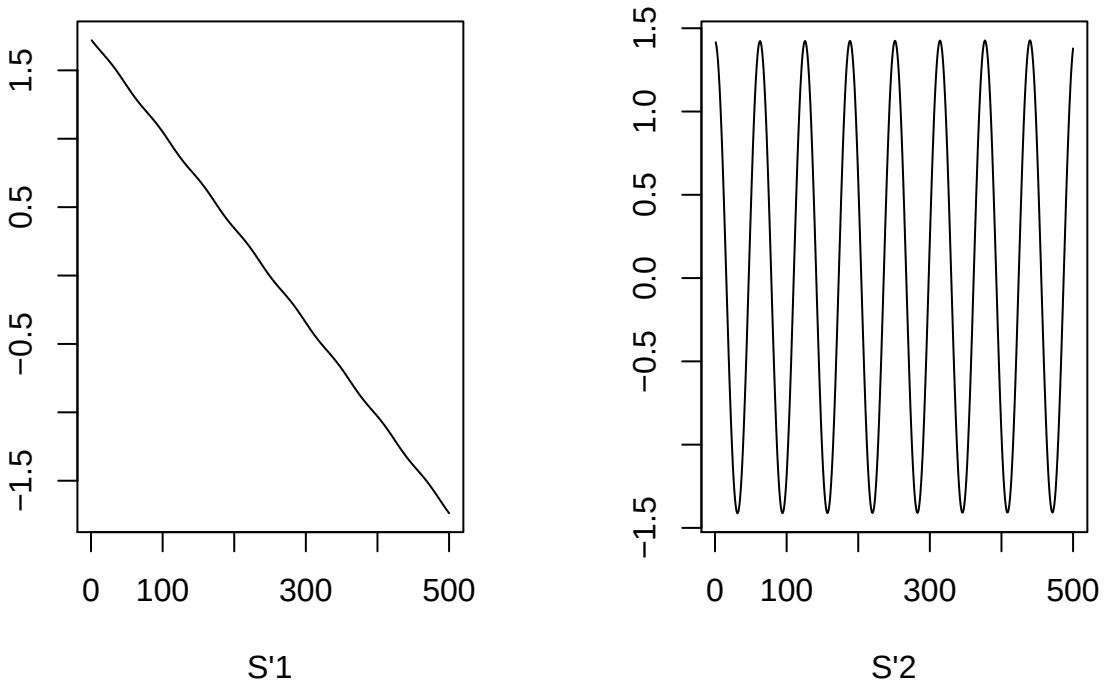
Mix S using A

```
X <- S %*% A
par(mfcol = c(1, 2))
plot(X[,1], type="l")
plot(X[,2], type="l")
```



Unmix using fastICA

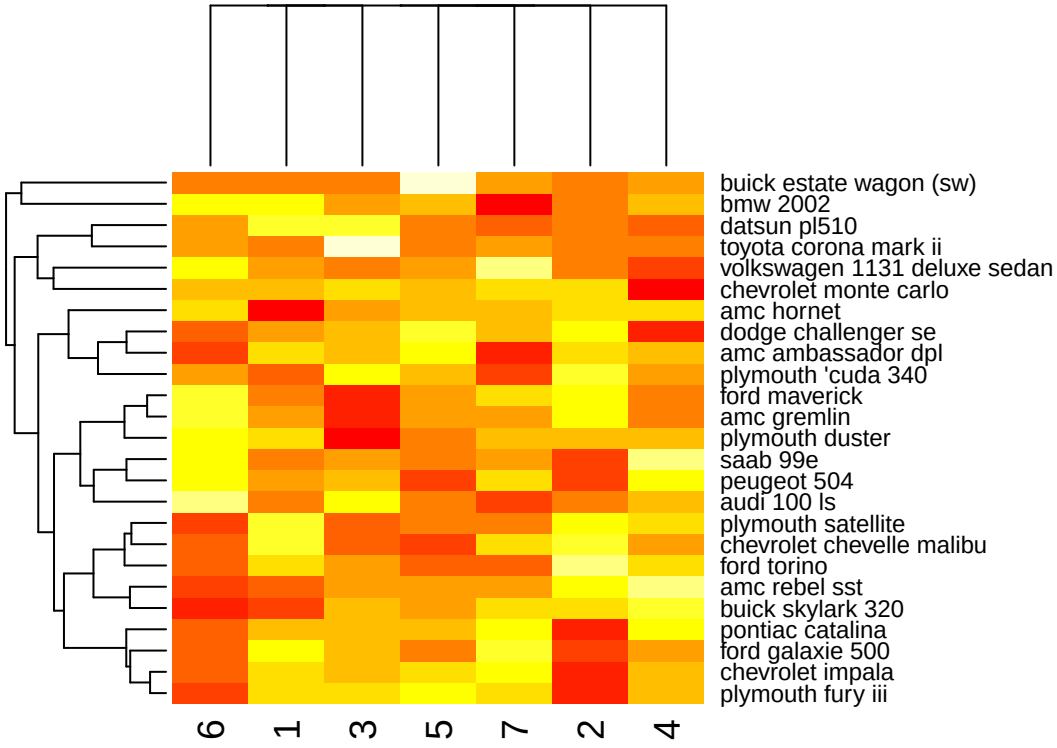
```
par(mfcol = c(1, 2))
plot(1:500, a$S[,1], type = "l", xlab = "S'1", ylab = "")
plot(1:500, a$S[,2], type = "l", xlab = "S'2", ylab = "")
```



ICA on the auto data

plot the independent components as a heatmap

```
heatmap(a$S)
```


buick estate wagon (sw)
bmw 2002
datsun pl510
toyota corona mark ii
volkswagen 1131 deluxe sedan
chevrolet monte carlo
amc hornet
dodge challenger se
amc ambassador dpl
plymouth 'cuda 340
ford maverick
amc gremlin
plymouth duster
saab 99e
peugeot 504
audi 100 ls
plymouth satellite
chevrolet chevelle malibu
ford torino
amc rebel sst
buick skylark 320
pontiac catalina
ford galaxie 500
chevrolet impala
plymouth fury iii
6
1
3
5
7
2
4

Non-negative Matrix Factorization

NMF is an algorithm that factorizes the given matrix into two matrices. All three matrices must have no negative values.

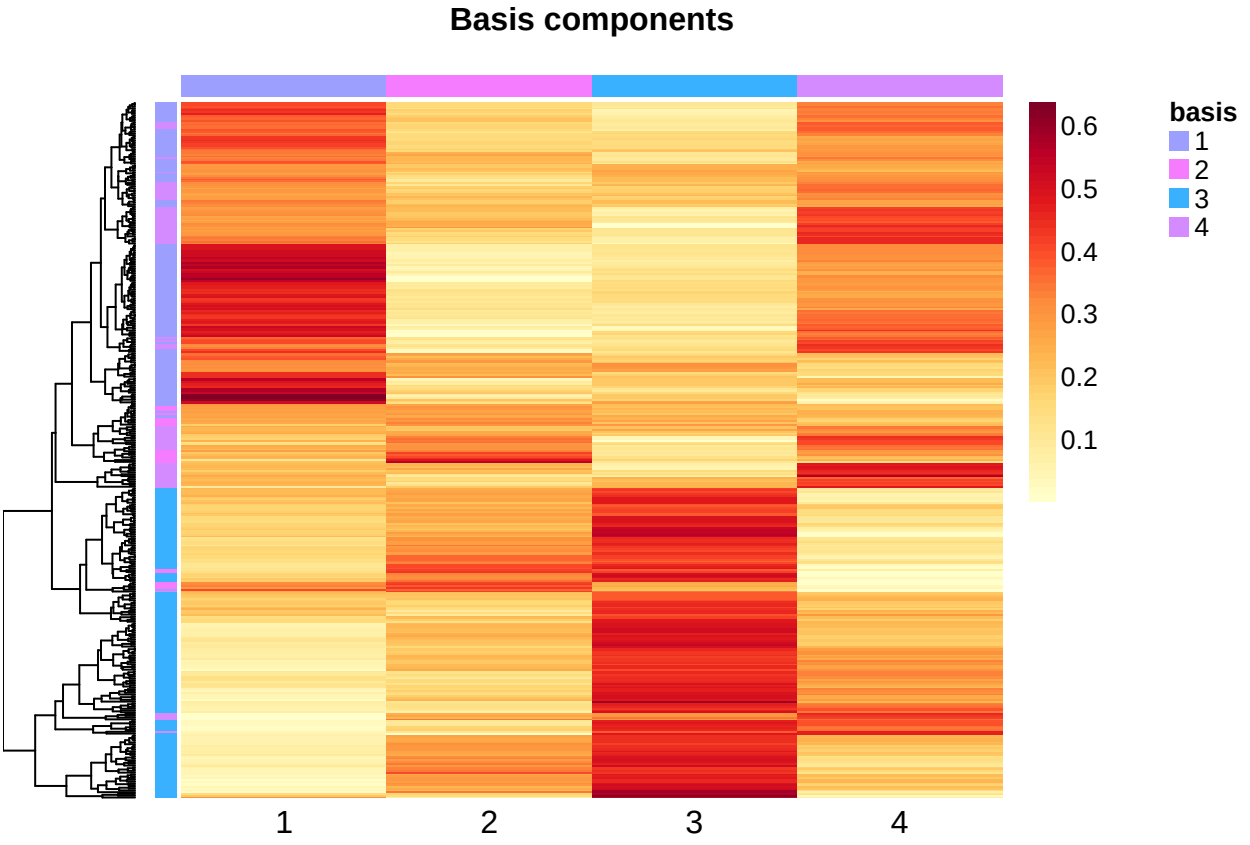
$$V_{mxn} = W_{mxp}H_{pxn}$$

Where p is specified to the algorithm. p can be thought of as the number of features to search for. The column vector W can be thought of as the features, and the vector H van be thought of as the weights for these features.

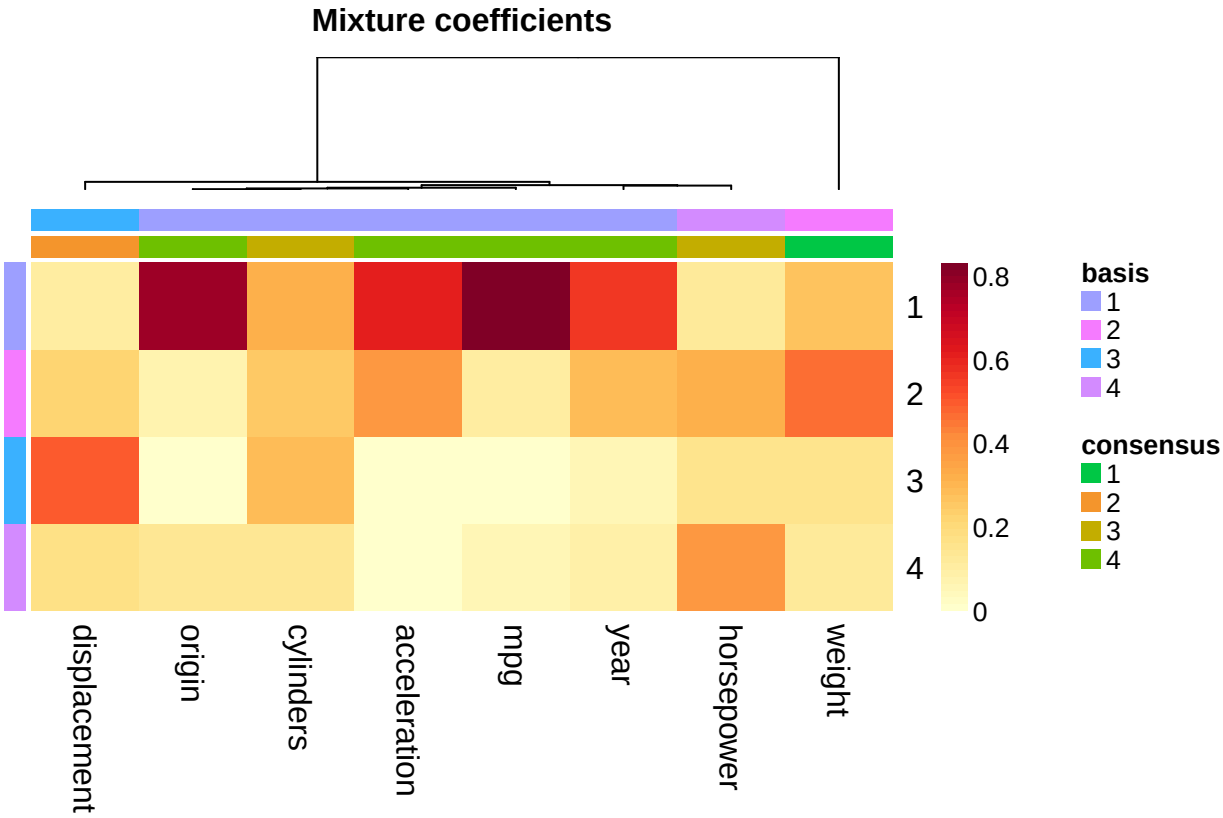
```
Auto_data_clust <- Auto_data[,1:8]

nmf_out <- nmf(Auto_data_clust, 4, set.seed(304543), nrun= 100)
W <- nmf_out@fit@W
H <- nmf_out@fit@H

#W
basismap(nmf_out)
```




```
#H
coefmap(nmf_out)
```


Mixture coefficients
1
2
3
4
0.8
0.6
0.4
0.2
0
basis
1
2
3
4
consensus
1
2
3
4
displacement
origin
cylinders
acceleration
mpg
year
horsepower
weight

###Homework

EXAMPLE CODE

```
# load the crab data
library(ggplot2)
library(MASS)
data(crabs)
crabs <- within(crabs,{
  levels(sp)  <- c('Blue','Orange')
  levels(sex) <- c('Female','Male')
})


# Perform PCA on the data
# retx returns the scores for each crab
# Append the components for each crab to the original data
crab.pca <- prcomp(crabs[,4:8],retx=TRUE)
crabs$PC1 <- crab.pca$x[,1]
crabs$PC2 <- crab.pca$x[,2]
crabs$PC3 <- crab.pca$x[,3]

# Print the rounded scores for the first three components
write.table(signif(crab.pca$rotation[,1:3],3),"rotations.tab")

#
# Plot discrimating factors found in the second component
#
second_component_dotplot <- ggplot(crabs, aes(y = CW, x = RW))        +
        geom_point()                                                  +
        scale_x_continuous("Rear width (mm)")     +
        scale_y_continuous("Carapace width (mm)")

png("second_component_dotplot.png"); print(second_component_dotplot)

#
# The second principle component corresponds to differences in
# the ratio of carapace width to rear width
#
second_component_density <- ggplot(crabs, aes(x = CW/RW))                                  +
        stat_density()                                                  +
        scale_x_continuous("Ratio of Carapace width to Rear width") +
        scale_y_continuous("Density")

png("second_component_density.png"); print(second_component_density)

#
# Discriminating factors for the third component
#
```

```
third_component_dotplot <- ggplot(crabs, aes(x = CW, y = BD))        +
        geom_point()                                  +
        scale_x_continuous("Carapace width (mm)") +
        scale_y_continuous("Body depth (mm)")

png("third_component_dotplot.png"); print(third_component_dotplot)

#
# The third component corresponds to body depth and carapace width
#
third_component_density <- ggplot(crabs, aes(x = BD/CW))                                  +
        stat_density()                                                  +
        scale_x_continuous("Ratio of Body Depth to Carapace Width") +
        scale_y_continuous("Density")

png("third_component_density.png"); print(third_component_density)


#
# Plot crab species for each of the two variables
#
morphology <- ggplot(crabs, aes(x = BD/CW, y = CW/RW, pch=sex, col=sp, size=sex)) +
        geom_point()                                                              +
        scale_x_continuous("Ratio of body depth to carapace width")         +
        scale_y_continuous("Ratio of carapace width to rear width")         +
        scale_shape_manual("Sex",values=c(1,15))                            +
        scale_color_manual("Crab\nSpecies",values=c("blue","orange3"))      +
        scale_size_manual(values=c(4,3),guide=FALSE)                        +
        theme(legend.position = "bottom")


png("morphology.png",height=600); print(morphology)

#
# Plot first two components
#
plot12 <- ggplot(crabs, aes(x = PC1, y = PC2, pch=sex, col=sp, size=sex)) +
        geom_point()                                                    +
        scale_x_continuous("First Principle Component")                 +
        scale_y_continuous("Second Principle Component")                +
        scale_shape_manual("Sex",values=c(1,15))                        +
        scale_color_manual("Crab\nSpecies",values=c("blue","orange3"))  +
        scale_size_manual(values=c(4,3),guide=FALSE)                    +
        theme(legend.position = "bottom")

png("first_components.png",height=600); print(plot12)
```

```
#
# Plot second two components
#
plot23 <- ggplot(crabs, aes(x = PC2, y = PC3, pch=sex, col=sp, size=sex)) +
        geom_point()                                                      +
        scale_x_continuous("Second Principle Component")                  +
        scale_y_continuous("Third Principle Component")                   +
        scale_shape_manual("Sex",values=c(1,15))                          +
        scale_color_manual("Crab\nSpecies",values=c("blue","orange3"))    +
        scale_size_manual(values=c(4,3),guide=FALSE)                      +
        theme(legend.position = "bottom")

png("second_components.png",height=600); print(plot23)
```

Load the functions with dataframe and covariance matrix

```
data(iris)
iris_subs <- iris[,c(1, 2, 3, 4)]
species <- iris[,5]
```

1. Run PCA, ICA, and NMF on the iris dataset.

a. Explain your inputs and outputs from each algorithm. For instance, in the input for the NMF example above, out inputs were a 25 x 8 matrix,and a rank of 4. The output was a matrix W with dimensions 25x4, and a matrix H with dimensions 4x8. We plotted the basis matrix (W), where each columns corresponds to a feature... etc.

2. Use the silhouette function in the cluster package to find the optimal number of clusters for kmeans for the iris dataset. Then cluster using hierarchical clustering and kmeans clustering. Does the data cluster by species?

Optional material

On PCA:

Eigen Vectors and Eigen Values http://www.visiondummy.com/2014/03/eigenvalues-eigenvectors/
Linear Algebra by Prof. Gilbert Strang https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/ http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenv

On ICA:

Independent Component Analysis: Algorithms and Applications https://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf Tutorial on ICA taken from http://rstudio-pubs-static.s3.amazonaws.com/93614_be30df613b2a4707b3e5a1a62f631d19.html