Unsupervised learning - clustering and dimension reduction

Brad Kim

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.

```r
setwd("/Users/MOOSE/Desktop/ML/Assignments/Lab2")
Auto_data <- read.table("Auto.data.txt", header=T, stringsAsFactors = F)
#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$horsepower)
#save car names
Auto_data_names <- Auto_data$name
#use the numeric values
Auto_data_clust <- Auto_data[,1:8]
dim(Auto_data_clust)
```

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

```r
#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]
```

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

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

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

```
#postitions 4 and 3 are the most similar
Auto_data_names[4]
```

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

```
Auto_data_names[3]
```

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

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.

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

Cluster Dendrogram

hierarchical_dist
hclust (*, "average")
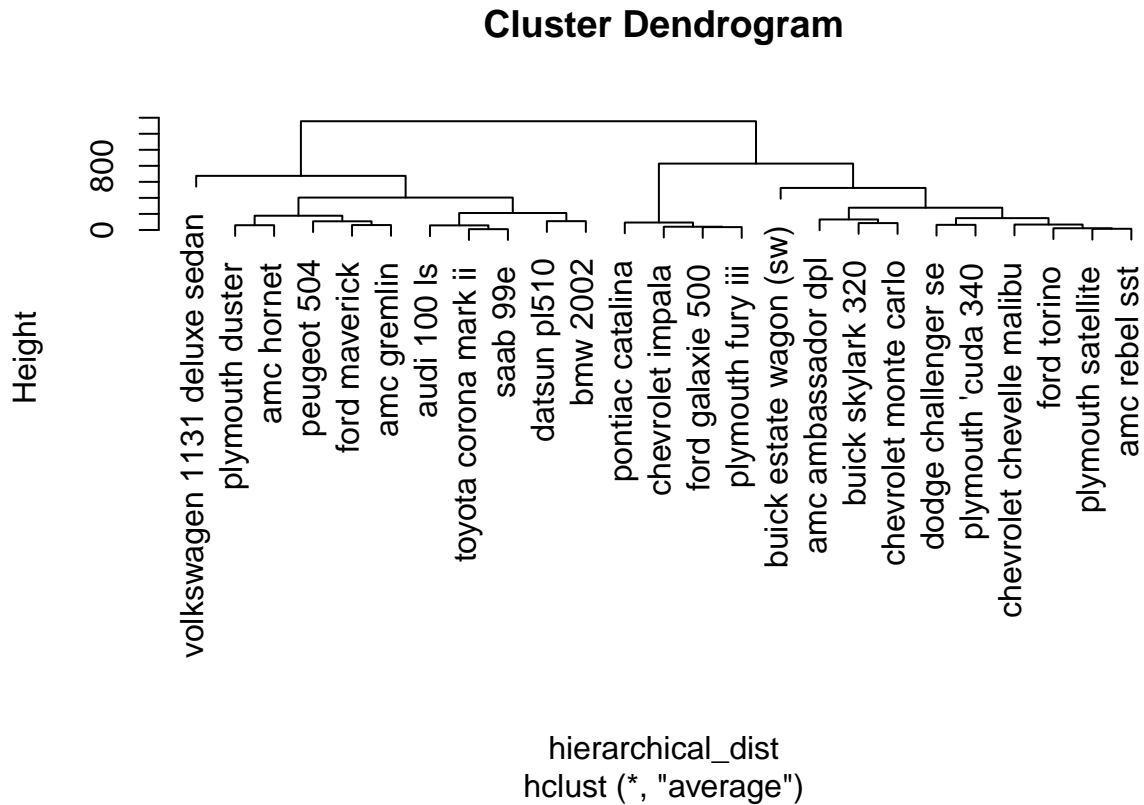
##K-Means Clustering Step 1. Choose the N number of clusters.

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(
    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?

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

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

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

```
##                               1
## chevrolet chevelle malibu 372.9148
```

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.

```
kmean_out <- kmeans(Auto_data_clust,2)
```

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.

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

```r
#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))
```

Step 2. Calculate covariance matrix of the Auto data

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

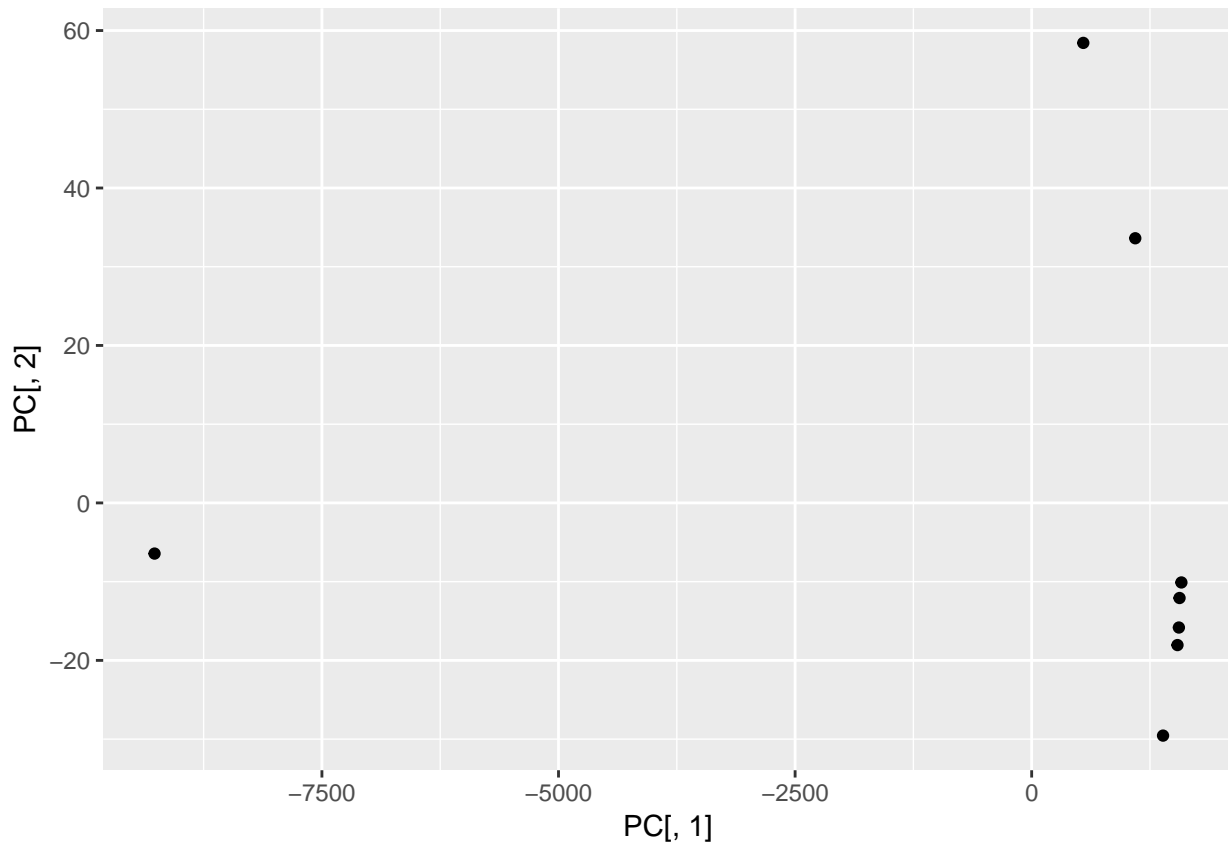Step 3. Calculate eigen values and vectors

```r
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.

```r
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]))
```

```
#+ geom_text(aes(label=Auto_data_names[1:8]), nudge_x = -2.5, nudge_y = 400)
```

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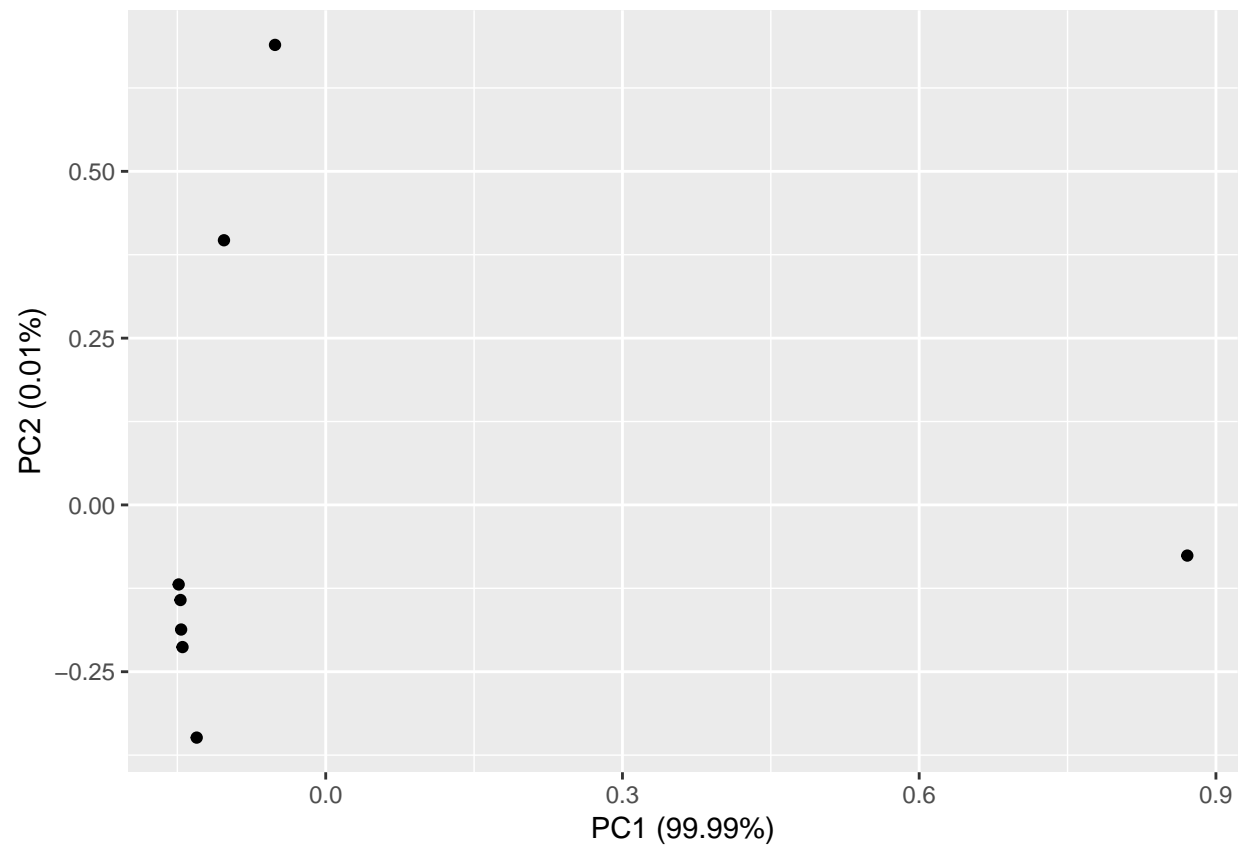
```
## [1]  99.99 100.00 100.00 100.00 100.00 100.00 100.00 100.00
```

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)))
```

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.

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

```r
#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")
```
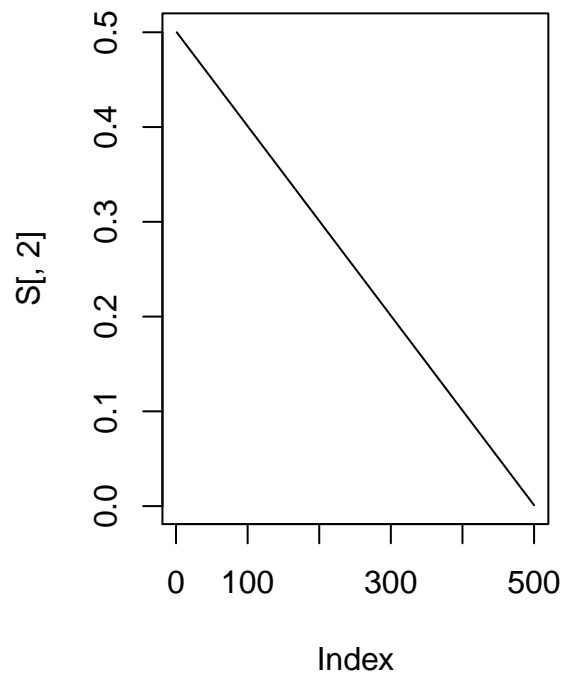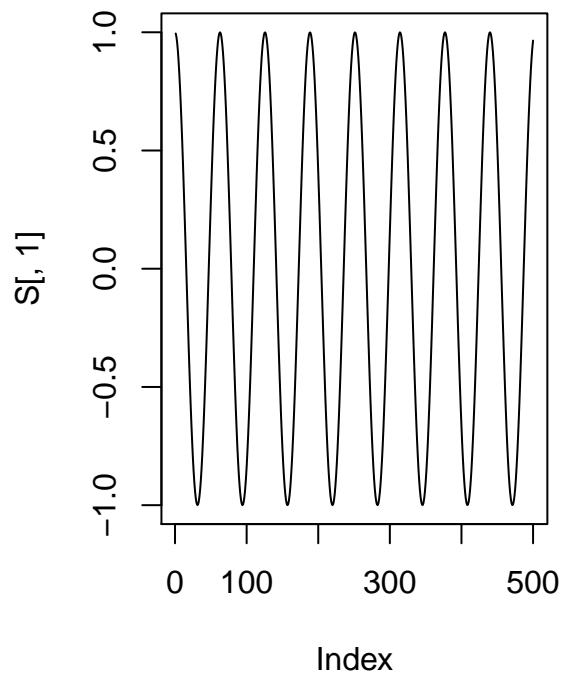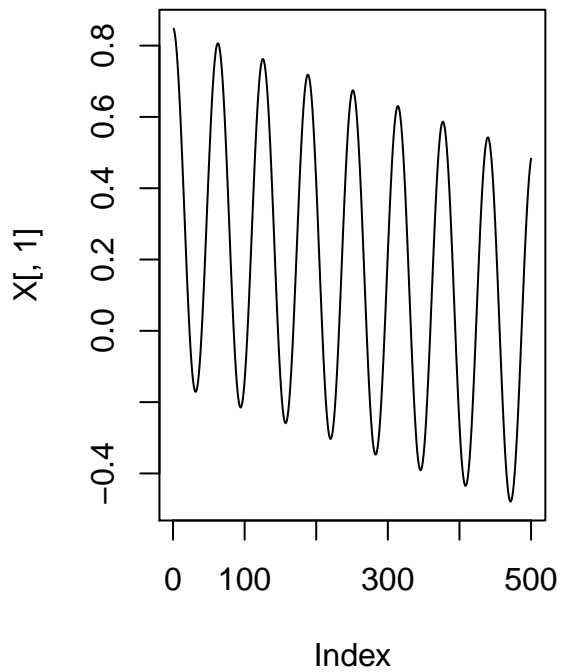

Create a mixing matrix A

```r
A <- matrix(c(0.5, 0.7, 0.423, 0.857), 2, 2)
```
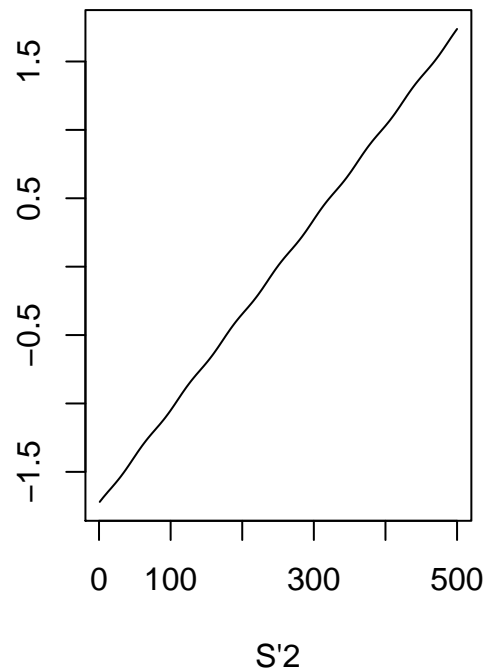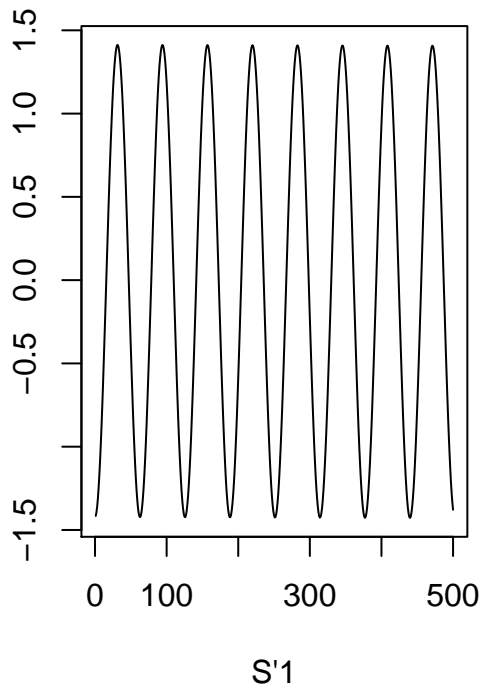
Mix S using A

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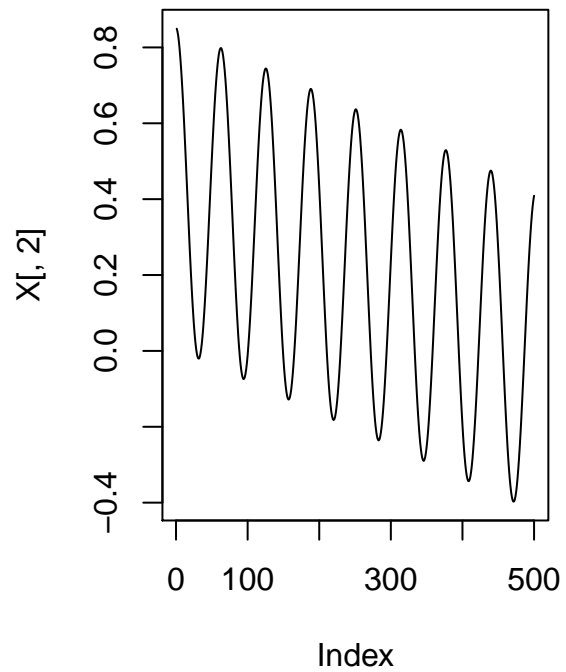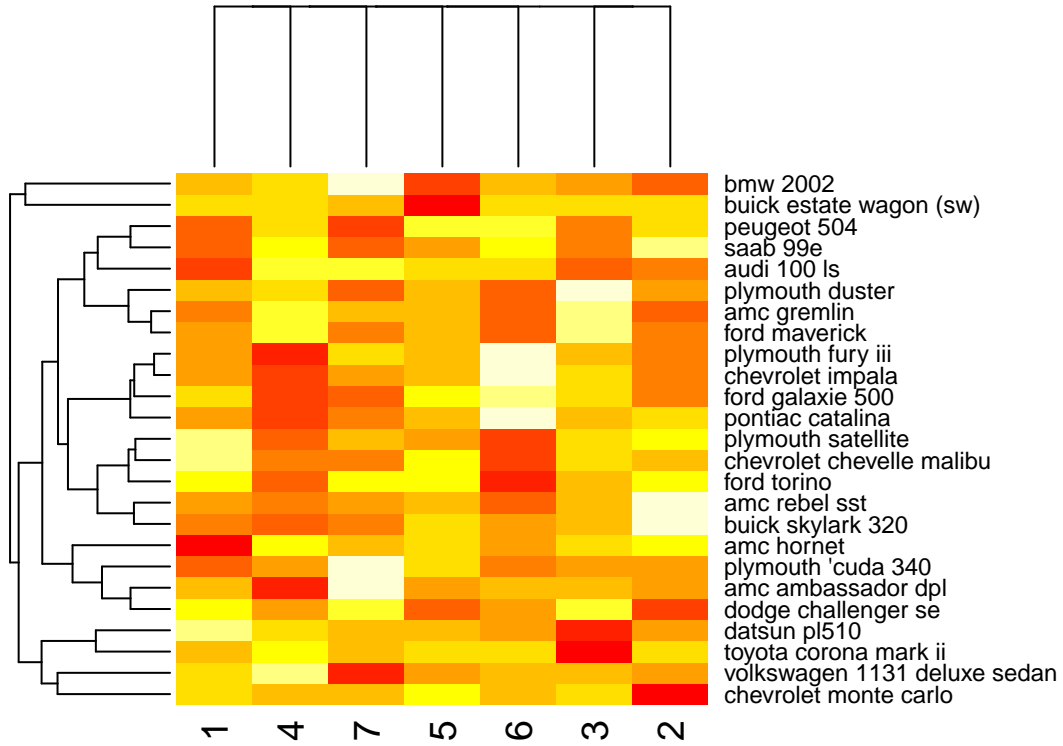
Unmix using fastICA

```
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

10

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

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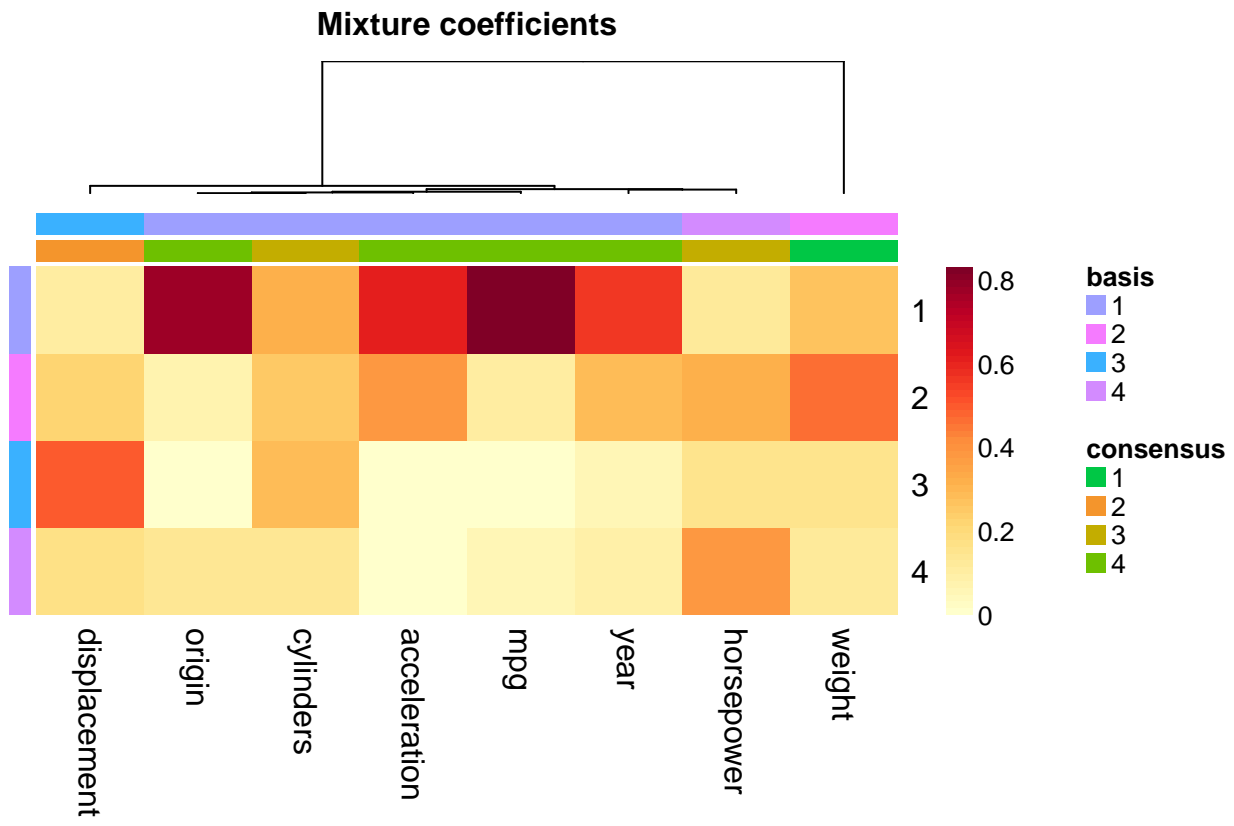
```r
Auto_data_clust <- Auto_data[,1:8]

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)
```

Basis components


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

Mixture coefficients
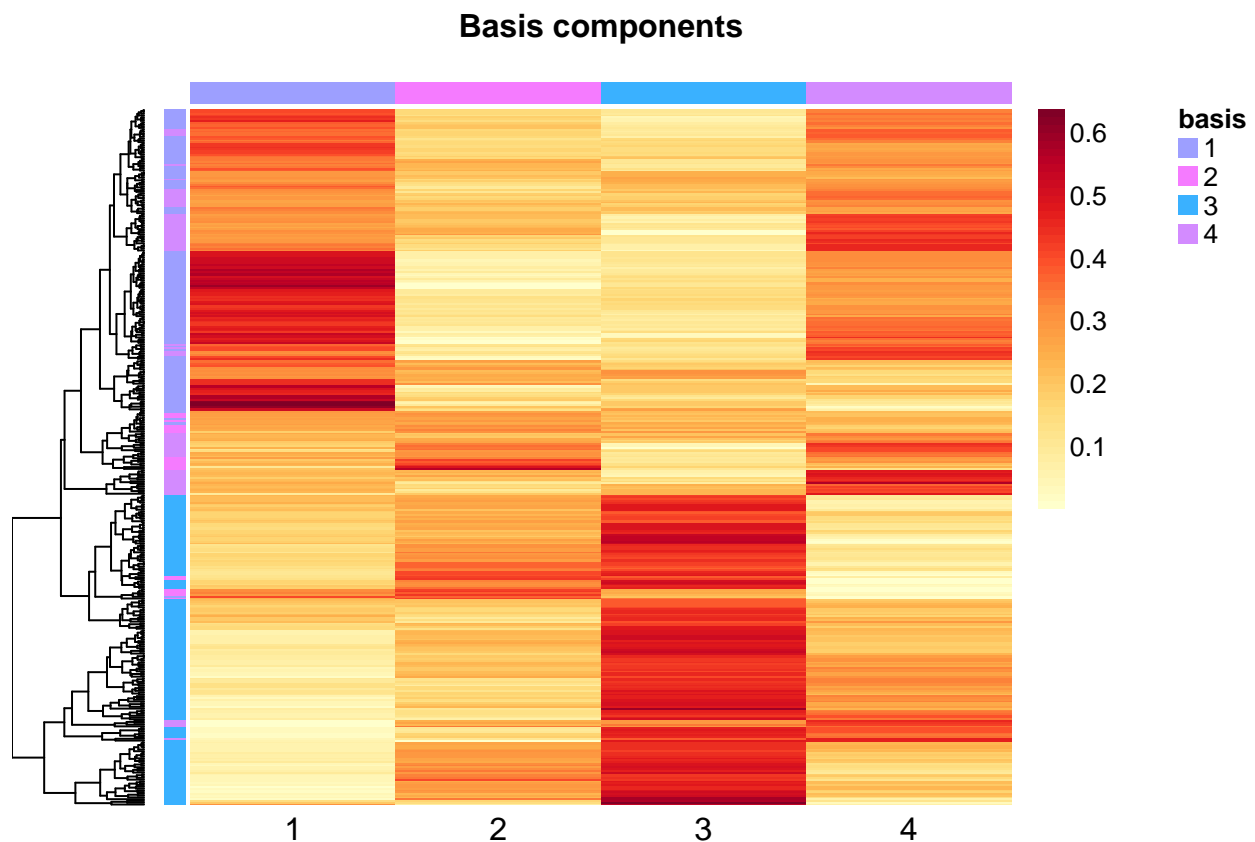
###Homework

```
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