Tarea 4

Estudiante

John Daniel hoyos Arias
Ivan Santiago Rojas Martinez
Genaro Alfonso Aristizabal Echeverri

Docente

Cesar Augusto Gomez Velez

Asignatura

Analitica de datos

Sede Medellín
29 de Noviembre del 2022

Índice

1. Ejercicio 1

Este ejercicio utiliza el conjunto de datos OJ el cual es parte de la librería ISLR

1.1. a) Cree un conjunto de entrenamiento con una muestra aleatoria de 800 observaciones y un conjunto de prueba que conste del resto de observaciones.

Se procede a cargar las librerías necesarias del **R** y a crear un conjunto de entrenamiento de **800** datos para prueba y **270** datos para entrenamiento fijando una semilla = **1** la cual permitirá la replicabilidad de nuestro informe.

```
require(ISLR)
require(tidyverse)
require(ggthemes)
require(caret)
require(e1071)
require(kableExtra)
```

```
set.seed(1)

data('OJ')
datos <- OJ

inTrain <- sample(nrow(OJ), 800, replace = FALSE)

training <- OJ[inTrain,]
testing <- OJ[-inTrain,]
```

1.2. b) Ajuste un clasificador de soporte vectorial utilizando cost = 0.1, con Purchase como la variable respuesta y las demás como predictores.

Tomando como variable respuesta **Purchase**. Se ajusta un clasificador de soporte vectorial lineal **(SVM Linear)** con un parámetro de **cost = 0.1**

```
svm_linear <- svm(Purchase ~ ., data = training,
                  kernel = 'linear',
                  cost = 0.1)
```

1.2.1. Utilice la función summary() para obtener un resumen de estadísticas y describa los resultados obtenidos.

```
summary(svm_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, kernel = "linear", cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  342
##
##  ( 171 171 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Este clasificador **SVM** de kernel **lineal** ha sido utilizado con **cost=0.1**, y se obtienen **342** vectores de soporte, **171** en una clase y **171** en la otra.

1.3. c) Que tasas de error de entrenamiento y de prueba obtiene?.

```
postResample(predict(svm_linear, training), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8350000 0.6532361
```

Se obtiene una tasa de precisión de clasificación del **83.5 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_linear, testing), testing$Purchase)
```

```
## Accuracy    Kappa
## 0.837037 0.640914
```

Se obtiene una tasa de precisión de clasificación del **83.7 %** para el conjunto de prueba.

Estos nos indica que este clasificador de soporte vectorial de kernel lineal tiene una alta capacidad predictiva.

1.4. d)Utilice la función tune() para obtener un valor óptimo del parámetro cost. Considere valores en el rango de 0.01 a 10.

```
set.seed(1)
svm_linear_tune <- train(Purchase ~ ., data = datos,
                         method = 'svmLinear2',
                         trControl = trainControl(method = 'cv', number = 10),
                         preProcess = c('center', 'scale'),
                         tuneGrid = expand.grid(cost = seq(0.01, 10, length.out = 20)))
```

```
## Support Vector Machines with Linear Kernel
##
## 1070 samples
##   17 predictor
##    2 classes: 'CH', 'MM'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 963, 963, 963, 963, 964, 963, ...
## Resampling results across tuning parameters:
##
##   cost        Accuracy   Kappa
##    0.0100000  0.8243018  0.6269416
##    0.5357895  0.8261540  0.6321140
##    1.0615789  0.8270798  0.6342259
##    1.5873684  0.8280318  0.6358094
##    2.1131579  0.8270973  0.6340117
##    2.6389474  0.8280406  0.6354812
##    3.1647368  0.8299098  0.6391508
##    3.6905263  0.8289752  0.6370765
##    4.2163158  0.8299098  0.6388080
##    4.7421053  0.8317791  0.6427817
##    5.2678947  0.8299100  0.6389136
##    5.7936842  0.8308359  0.6410463
##    6.3194737  0.8308359  0.6410463
```

```
##    6.8452632  0.8317705  0.6428938
##    7.3710526  0.8317705  0.6428938
##    7.8968421  0.8317705  0.6428938
##    8.4226316  0.8326964  0.6449667
##    8.9484211  0.8317618  0.6431706
##    9.4742105  0.8317618  0.6431706
##   10.0000000  0.8317618  0.6431706
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cost = 8.422632.
```

El valor optimo para el parámetro **cost** usando como criterio de selección el **acurracy** es **cost = 8.4226316** con una precisión de **83.26 %** de clasificación.

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  2.01
##
## - best performance: 0.1626168
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.1691589 0.04024604
## 2  1.01 0.1635514 0.03872213
## 3  2.01 0.1626168 0.03795001
## 4  3.01 0.1626168 0.03994348
## 5  4.01 0.1635514 0.04067777
## 6  5.01 0.1654206 0.03917066
## 7  6.01 0.1654206 0.03917066
## 8  7.01 0.1672897 0.03671522
## 9  8.01 0.1672897 0.03671522
## 10 9.01 0.1682243 0.03606180
```

1.5. e) Calcule nuevamente las tasas de error de entrenamiento y de prueba usando el valor óptimo obtenido de cost.

```
postResample(predict(svm_linear_tune, training), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8375000 0.6581084
```

Se obtiene una tasa de precisión de clasificación del **83.4 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_linear_tune, testing), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.8444444 0.6585983
```

Se obtiene una tasa de precisión de clasificación del **84.8 %** para el conjunto de prueba.

El modelo encontrado por medio de validación cruzada donde se buscaba encontrar el valor más óptimo para **cost** con un valor de **00.1** tiene una leve mejoría respecto al primer modelo con un **cost = 0.1** si son comparados por medio del **accuracy**.

1.5.1. Repita items de (b) hasta (e) ajustando esta vez una máquina de soporte vectorial (svm) con un nucle radial. Utilizando el valor de default paray

Ajustamos un clasificador de soporte vectorial de kernel radial **(SVM Radial)** con un parámetro de **cost = 0.1**

```
svm_radial <- svm(Purchase ~ ., data = training,
                  method = 'radial',
                  cost = 0.1)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, method = "radial", cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
## Number of Support Vectors:  541
##
##  ( 272 269 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Este clasificador **SVM** de kernel **Radial** ha sido utilizado con **cost=0.1**, y se obtienen **541** vectores de soporte, **272** en una clase y **269** en la otra.

```
postResample(predict(svm_radial, training), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8262500 0.6288385
```

Se obtiene una tasa de precisión de clasificación del **82.63 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_radial, testing), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.7962963 0.5502453
```

Se obtiene una tasa de precisión de clasificación del **79.62 %** para el conjunto de prueba.

Comparando la tasa de precisión del clasificador de maquina de soporte vectorial con kernel lineal y el clasificador de maquina de soporte vectorial con kernel radial se obtiene una perdida de precisión si se usa el **SVM Radial**.

Procedemos a buscar un valor optimo para **cost**

```
set.seed(1)
svm_radial_tune <- train(Purchase ~ ., data = training,
                         method = 'svmRadial',
                         trControl = trainControl(method = 'cv', number = 10),
                         preProcess = c('center', 'scale'),
                         tuneGrid = expand.grid(C = seq(0.01, 10, length.out = 20),
                                                sigma = 0.05))
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 800 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
```

```
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 720, 721, 721, 719, 719, 721, ...
## Resampling results across tuning parameters:
##
##   C          Accuracy   Kappa
##    0.0100000  0.6062600  0.0000000
##    0.5357895  0.8212641  0.6208307
##    1.0615789  0.8262648  0.6308805
##    1.5873684  0.8224990  0.6229775
##    2.1131579  0.8262807  0.6311956
##    2.6389474  0.8199986  0.6168480
##    3.1647368  0.8175141  0.6113156
##    3.6905263  0.8162641  0.6084492
##    4.2163158  0.8187799  0.6138051
##    4.7421053  0.8162953  0.6084718
##    5.2678947  0.8150141  0.6050869
##    5.7936842  0.8174832  0.6100076
##    6.3194737  0.8162486  0.6075335
##    6.8452632  0.8149828  0.6051077
##    7.3710526  0.8149828  0.6051077
##    7.8968421  0.8137482  0.6026385
##    8.4226316  0.8137482  0.6026385
##    8.9484211  0.8124982  0.6001922
##    9.4742105  0.8099666  0.5947747
##   10.0000000  0.8099666  0.5953119
##
## Tuning parameter 'sigma' was held constant at a value of 0.05
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.05 and C = 2.113158.
```

El valor optimo para el parámetro **cost** usando como criterio de selección el **acurracy** es **cost = 1.0615789** con una precisión de **82.62 %**.

```
postResample(predict(svm_radial_tune, training), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8525000 0.6865141
```

Se obtiene una tasa de precisión de clasificación del **85.25 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_radial_tune, testing), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.8185185 0.6009050
```

Se obtiene una tasa de precisión de clasificación del **81.85 %** para el conjunto de prueba.

1.5.2. Repita items (b) hasta (e) utilizando nuevamente una máquina de soporte vectorial pero esta vez con un nucleo polinomial, usando degree = 2.

Ajustamos un clasificador de soporte vectorial de kernel no lineal **(SVM Polynomial)** con un parámetro de **cost = 0.1**

```
svm_poly <- svm(Purchase ~ ., data = training,
                  method = 'polynomial', degree = 2,
                  cost = 0.01)
summary(svm_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = training, method = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  634
##
##  ( 319 315 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Este clasificador **SVM** de kernel **Polynomial** ha sido utilizado con **cost=0.1**, y se obtienen **634** vectores de soporte, **319** en una clase y **315** en la otra.

```
postResample(predict(svm_poly, training), training$Purchase)
```

```
## Accuracy    Kappa
##  0.60625  0.00000
```

Se obtiene una tasa de precisión de clasificación del **60.62 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_poly, testing), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.6222222 0.0000000
```

Se obtiene una tasa de precisión de clasificación del **62.22 %** para el conjunto de prueba.

```
set.seed(2)
svm_poly_tune <- train(Purchase ~ ., data = training,
                       method = 'svmPoly',
                       trControl = trainControl(method = 'cv', number = 10),
                       preProcess = c('center', 'scale'),
                       tuneGrid = expand.grid(degree = 2,
                                              C = seq(0.01, 10, length.out = 20),
                                              scale = TRUE))
```

```
## Support Vector Machines with Polynomial Kernel
##
## 800 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 720, 720, 720, 720, 720, 720, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##    0.0100000  0.8186828  0.6083061
##    0.5357895  0.8149633  0.6073699
##    1.0615789  0.8174949  0.6125370
##    1.5873684  0.8174637  0.6118817
##    2.1131579  0.8174637  0.6118824
```

```
##    2.6389474  0.8199795  0.6167680
##    3.1647368  0.8199795  0.6166926
##    3.6905263  0.8162295  0.6080579
##    4.2163158  0.8149949  0.6062309
##    4.7421053  0.8187295  0.6144184
##    5.2678947  0.8224795  0.6225828
##    5.7936842  0.8187295  0.6140408
##    6.3194737  0.8199795  0.6170933
##    6.8452632  0.8174949  0.6121932
##    7.3710526  0.8162291  0.6097369
##    7.8968421  0.8137291  0.6038804
##    8.4226316  0.8149791  0.6067533
##    8.9484211  0.8137291  0.6038804
##    9.4742105  0.8124945  0.6014315
##   10.0000000  0.8137445  0.6044150
##
## Tuning parameter 'degree' was held constant at a value of 2
## Tuning
##  parameter 'scale' was held constant at a value of TRUE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 2, scale = TRUE and C
##  = 5.267895.
```

El valor optimo para el parámetro **cost** usando como criterio de selección el **acurracy** es **cost = 2.6389474** con una precisión de **81.99 %**.

```
postResample(predict(svm_poly_tune, training), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8625000 0.7074371
```

Se obtiene una tasa de precisión de clasificación del **86.25 %** para el conjunto de entrenamiento.

```
postResample(predict(svm_poly_tune, testing), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.8148148 0.5886654
```

Se obtiene una tasa de precisión de clasificación del **81.48 %** para el conjunto de prueba.

1.6. h) En general cúal método parece proporcionar los mejores resultados en estos datos?.

Kernel	Best Cost	Acurracy-Conjunto de prueba
Lineal	0.01	84.81 %
Radial	1.0615789	81.85 %
Polynomial	2.6389474	81.48 %

En general, los modelos son muy similares en el momento de hacer predicciones, resaltamos el núcleo lineal, ya que es el modelo con mayor precisión. Los clasificadores de soporte vectorial de kernel radial y polynomial tiene perdida de precisión por un pequeño margen.

2. Ejercicio 2

```
# Cargo de librerias
library(corrplot)
```

Se considera el conjunto de datos **USArrests**. En este ejercicio se agruparán los estados en **USArrests** con agrupamiento jerarquico. Este conjunto de datos contiene estadísticas, en arrestos por cada 100,000 residentes por agresión, asesinato y violación en cada uno de los 50 estados de EE. UU. en 1973. También se proporciona el porcentaje de la población que vive en áreas urbanas.

Primeramente, se procede a cargar la base de datos **USArrests** y examinar sus características:

```
head(USArrests) # encabezado de la base
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
str(USArrests) # caracteristicas de la base
```

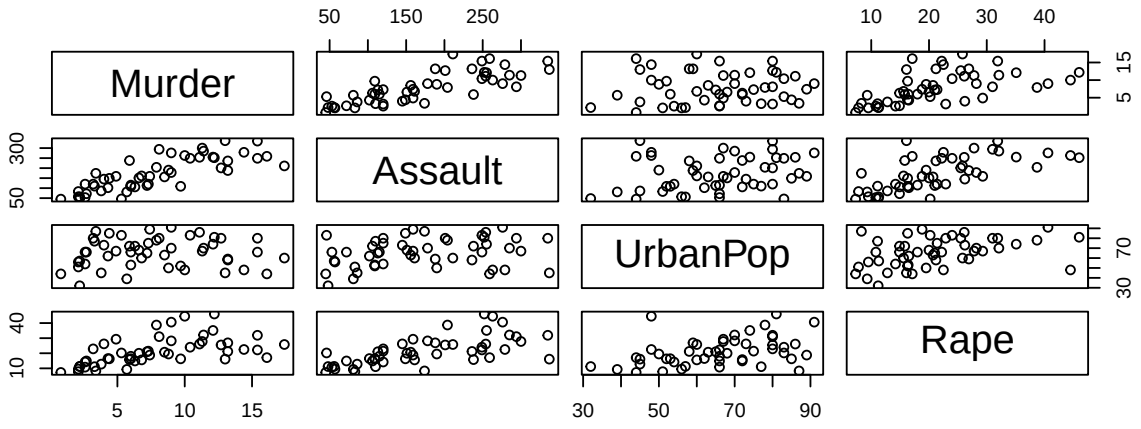
```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

Se observa como la base cuenta con 50 observaciones y 4 variables las cuales todas son numericas y su descripción se presenta seguidamente:

- **Murder:** Arrestos por asesinato (por 100.000).
- **Assault:** Arrestos por asalto (por 100.000).
- **UrbanPop:** Porcentaje de población urbana.
- **Rape:** Arrestos por violaciones (por 100.000).

Adicionalmente, se presentan un análisis descriptivos de estas variables:

```
pairs(USArrests)
```



Del anterior gráfico de dispersión entre las variables, se observa como entre cada par de combinación de variables, existe una relación creciente. Lo cual en primera instancia podría ser un indicativo de que posiblemente en los estados donde se presente mayor porcentaje de población urbana también se puede presentar mayores casos de arrestos por asalto, asesinato o violación.

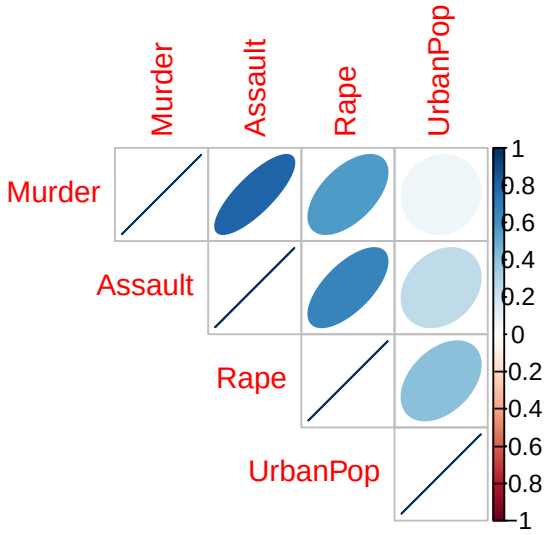
Por otro lado, se presenta una matriz de correlación entre las cuatro variables:

```
cor(USArrests)
```

```
##              Murder   Assault   UrbanPop      Rape
## Murder   1.00000000 0.8018733 0.06957262 0.5635788
## Assault  0.80187331 1.0000000 0.25887170 0.6652412
## UrbanPop 0.06957262 0.2588717 1.00000000 0.4113412
## Rape     0.56357883 0.6652412 0.41134124 1.0000000
```

También, se presenta un gráfico de correlaciones de estas variables:

```
# se crea la matriz de correlación
corr.data <- cor(USArrests)
# gráfico de correlacion para las variables cuantitativas
corrplot(corr.data, method = 'ellipse', order='AOE', type = 'upper')
```



De los resultados obtenidso anteriormente, se observa como:

- Existe una alta correlación positiva entre los arrestos por asesinato y los arrestos por asaltos, la cual es de un 0.8018733, Esto puede indicar que, así como pueden aumentar los arrestos por asalto en un estado de USA, también puede aumentar los arrestos por asesinato en ese mismo estado.
- Existe una alta correlación positiva entre los arrestos por asalto y los arrestos por violaciones, la cual es de un 0.6652412, Esto puede indicar que, así como pueden aumentar los arrestos por asalto en un estado de USA, también puede aumentar los arrestos por violaciones en ese mismo estado.
- Se observa en la matriz de correlaciones como, no existe una aparente correlación significativa entre los arrestos por asesinatos y el porcentaje de población urbana.

2.1. a) Utilice agrupación jerárquica con enlace completo y distancia euclidiana,

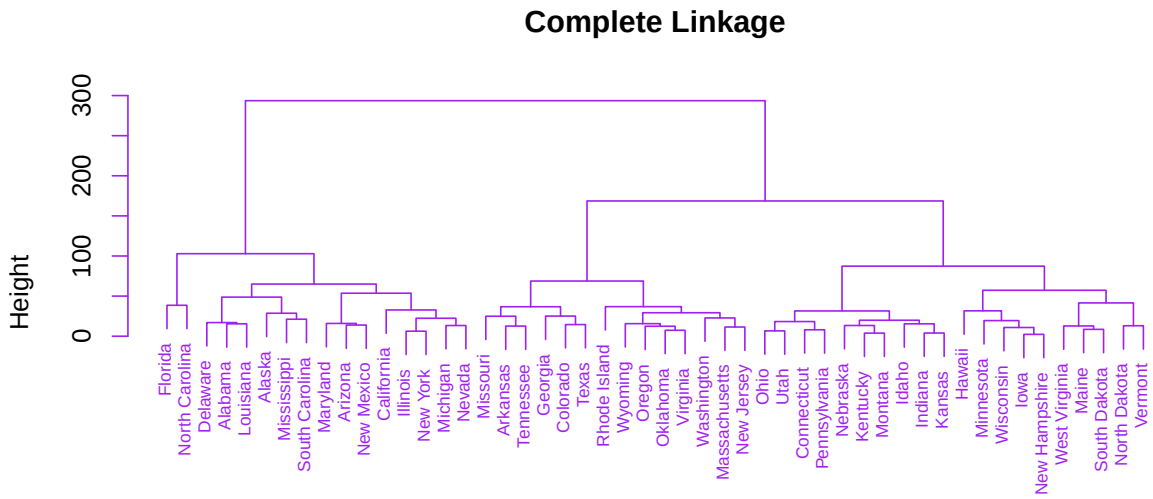
para agrupar los estados.

Se utiliza agrupación jerárquica con enlace completo y distancia euclidiana, para agrupar los estados, de la siguiente forma:

```
# ajustando un enlace completo con la distancia euclidiana
hc_complete =hclust(dist(USArrests, method = "euclidean"), method ="complete")
```

Luego se presenta el **dendrograma** de dicho enlace completo:

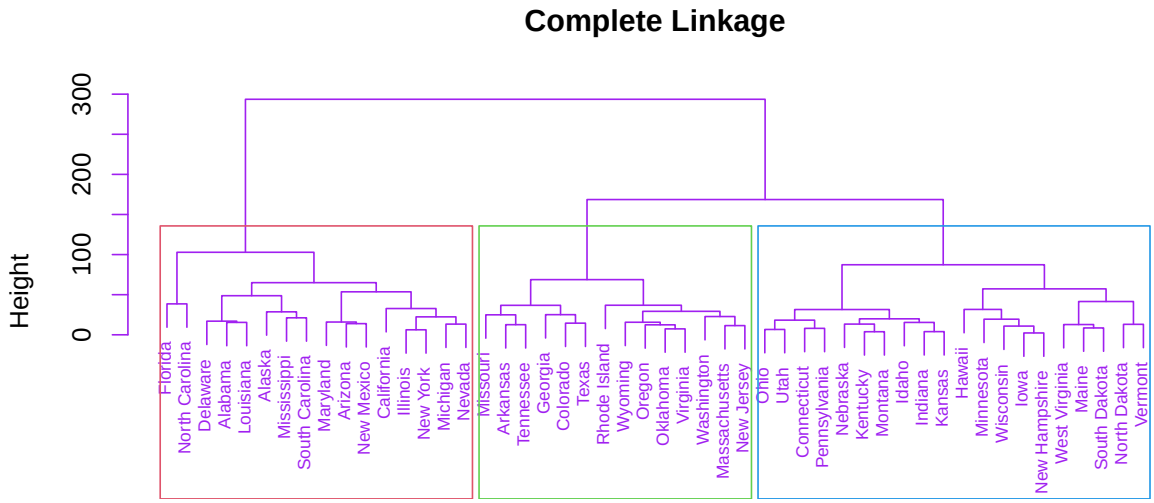
```
plot(hc_complete ,main =" Complete Linkage ", xlab="", sub ="",
cex =.7, col = "Purple")
```



2.2. b) Corte el dendograma a una altura que dé como resultado tres clusters. Qué estados pertenecen a qué cluster?

Se procede a separar en el **dendograma** a una altura que dé como resultado 3 *clusters*.

```
plot(hc_complete ,main =" Complete Linkage ", xlab="", sub ="",
cex =.7, col = "Purple")
rect.hclust(hc_complete, k=3, border=2:10)
```



Luego, usando la función **cutree()** se puede observar las etiquetas a las que pertenece cada estado según el cluster al que se le asigno.

```
cutree (hc_complete, 3)
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              3              1              1              2
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              3              1              3              3
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              3              3              1              3              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              3              1              2
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              3              3              1              3              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              3              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              3              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              3              2              2              3              3
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              3              3              2
```

2.3. c) Agrupe jerárquicamente los estados utilizando un enlace completo y distancia euclidiana, después de escalar las variables para tener una desviación estándar uno.

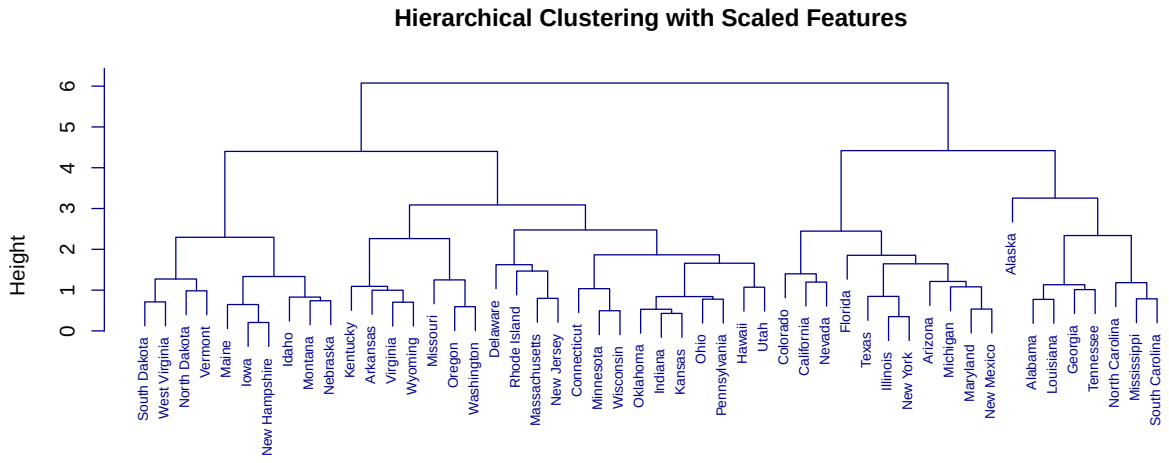
Ahora se procede a escalar las variables a fin de tener una desviación estándar de uno y luego se realiza la respectiva agrupación jerárquica usando un enlace completo y la distancia euclidiana.

La función **scale()** permite escalar las variables, así como el proceso de agrupación jerárquica se muestra a continuación:

```
# ajustando un enlace completo con la distancia euclidiana y las variables escaladas
hc_complete_scale =hclust(dist(scale(USArrests), method = "euclidean"),
                          method ="complete")
```

Luego se presenta el **dendrograma** de dicho enlace completo:

```
plot(hc_complete_scale ,main ="Hierarchical Clustering with Scaled Features",
     xlab="", sub ="",
cex =.7, col = "dark blue")
```

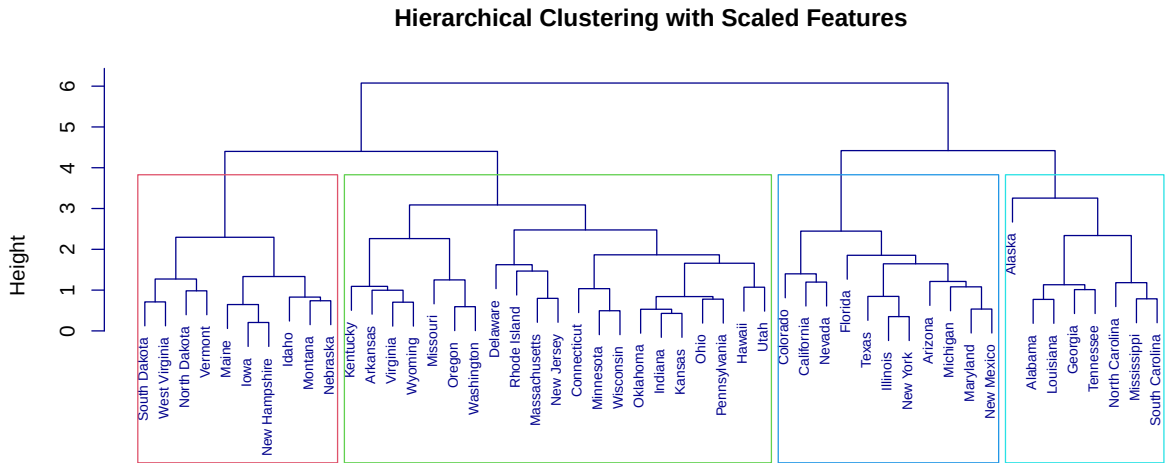


2.4. d)¿Qué efecto tiene el escalado de las variables en la estructura jerárquica del agrupamiento obtenido? En su opinión, ¿deberían las variables ser escaladas antes de que se calculen las disimilitudes entre observaciones? Proporcione una justificación para su respuesta.

Se observa como, en el **dendrograma** correspondiente a las agrupaciones jerárquicas con las variables escaladas, es notablemente distinto a la agrupación jerárquica generada sin las variables escaladas. Dado que, si bien estamos tratando con los mismos datos, la escalación de variables hace que en cada sub rama existan agrupaciones más uniformes. Inicialmente con las variables sin escalar se observaba claramente una distinción entre tres grupos o *clusters* distintos. En cambio, realizando el escalado de las variables se puede observar una posible distinción entre 4 *clusters*.

A continuación se presenta una posible agrupación entre 4 *clusters*:

```
plot(hc_complete_scale ,main ="Hierarchical Clustering with Scaled Features",
     xlab="", sub ="",
cex =.7, col = "dark blue")
rect.hclust(hc_complete_scale, k=4, border=2:10)
```



Aunque también podría ser una agrupación de tres *clusters*.

En definitiva, se considera que las variables deben ser escaladas previamente, ya que proporciona una mejor estabilidad a la hora de hacer agrupaciones jerárquicas. Porque es bien sabido que la distancia euclidiana no tiene en cuenta el tipo de escala en la cual se encuentran

las variables. Lo cual hace que, en ocasiones, usar esta medida no sea del todo preciso y como se pudo observar en los literales anteriores, se obtuvieron *clusters* y **dendrogramas** distintos.