Tarea 1_Modulo3

Luisa María Acosta O. **Laura Camila Agudelo O.** **Karen Andrea Amaya M.**

noviembre 2020

1. Librerías

```
#Librerias
library(kableExtra) #tablas
library(ISLR)
library(tree)
library(randomForest)
```

1.

- Nota: Posiblemente al ejecutar el script se puedan observar resultados distintos a los presentados en este documento para el literal **1**, esto, debido a un error entre la compilación al latex y el Rstudio. Aunque esto puede resolverse si se ejecuta en Rstudio cloud y allí sí se observarán los resultados iguales a los expuestos aquí.

a) Datos:

La base de datos "Carseats" se dividirá en un conjunto de entrenamiento y prueba de proporciones 70 y 30 respectivamente, pues empiricamente se ha comprobado que dicha partición genera buenos resultados.

```
set.seed(1)
n <- length(Carseats$Sales)
train<- sample(n,(n*0.7))
```

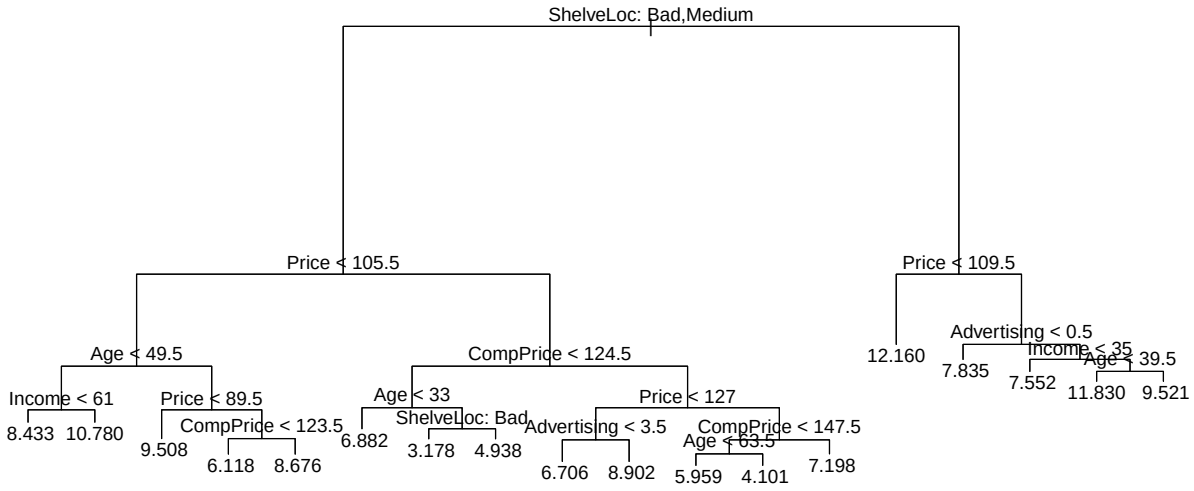
b) Ajuste para arbol de regresión:

- Arbol de regresión:

```
arbol = tree(Sales~.,Carseats, subset = train)
summary(arbol)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats, subset = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Income"      "CompPrice"
## [6] "Advertising"
## Number of terminal nodes:  18
## Residual mean deviance:  2.409 = 631.1 / 262
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.77800 -0.96100 -0.08865  0.00000  1.01800  4.14100
```

```
plot(arbol)
text(arbol,pretty=0)
```



Se observa que para el árbol de regresión las variables necesarias para la prediccion de la variable "Sales" son:"ShelveLoc", "Price", "Age", "Income", "CompPrice" y "Advertising". Y podemos decir que el indicador más importante para la preducción de la variable "Sales", puede ser "ShelvecLoc" (calidad de la ubicacion de las estanterias de los asientos), puesto que en la primera rama se separa la categoría "Good" de las categorías "Bad" y "Medium".

Además, según el árbol las ventas en promedio serán mas altas cuando la calidad de ubicación es buena y el precio es menor a 109.5. Por otro lado, las ventas en promedio más bajas se darán cuando la calidad de ubicación es mala, el precio es mayor a 105.5, el precio de la competencia es menor a 124.5 y la edad media de la población local es mayor a 33.

- MSE:

```
yhat = predict(arbol ,newdata = Carseats[-train ,])
arbol.test = Carseats[-train ,"Sales"]
mean((yhat-arbol.test)^2)
```

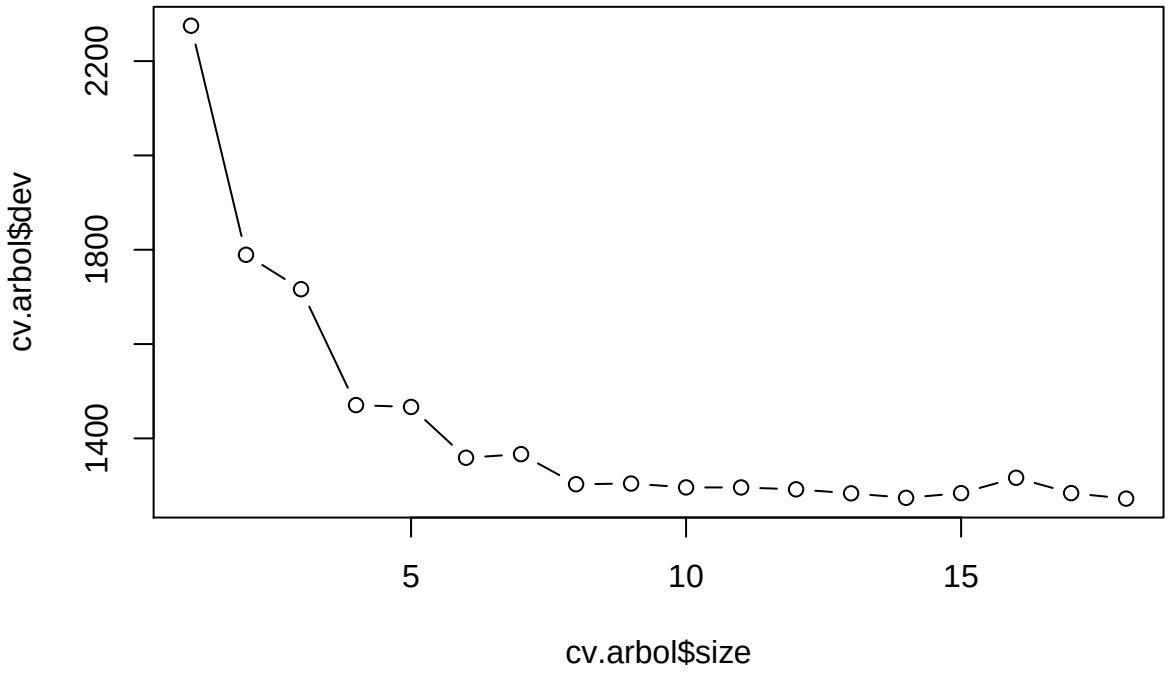
```
## [1] 4.208383
```

El MSE en el conjunto de prueba asociado con el árbol de regresión es de 4.2084. La raiz cuadrada de 4.2084 es 2.05 ó aproximadamente 2. Indicando que en este modelo las predicciones están alrededor de 2000 de la verdadera cantidad de ventas en cada tienda.

c) Validació cruzada:

Utilizaremos la función cv.tree() para ver si una poda del árbol creado en el literal b) mejora su desempeño.

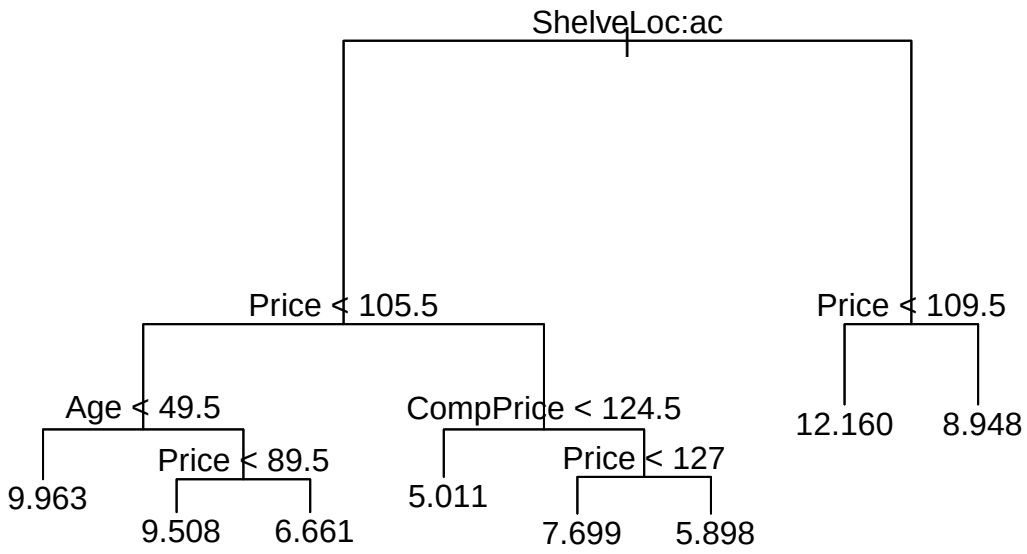
```
cv.arbol =cv.tree(arbol)
plot(cv.arbol$size ,cv.arbol$dev,type="b")
```



Se observa que 8 nodos es una candidad considerable para reducir el error de validación de una forma considerable, evitando sobreajustar el modelo con un mayor numero de nodos. Por lo cual, se podará el arbol con dicha cantidad.

```
prune.carseats<-prune.tree(arbol,best=8)

plot(prune.carseats)
text(prune.carseats)
```




```
yphat=predict(prune.carseats,Carseats[-train,])
arbolp.test = Carseats[-train ,"Sales"]
mean((yphat-arbolp.test)^2)
```

```
## [1] 4.579256
```

Y con esto, se puede observar que haber podado el árbol no mejora el MSE de prueba del modelo.

d) Bagging:

Ahora se aplicará Bagging al conjunto de datos Carseats, utilizando el paquete randomForest.

```
set.seed(1)
arbolbag<-randomForest(Sales~.,data=Carseats,subset=train,mtry=10,importance=T)

ybhat<-predict(arbolbag,Carseats[-train,])
mean((ybhat-Carseats[-train,"Sales"])^2)
```

```
## [1] 2.573252
```

Podemos observar que el MSE de prueba asociado con el modelo Bagged es de 2.573, un valor mucho menor que el logrado en el literal anterior, logrando reducir el MSE a casi la mitad del MSE de prueba hallado con el árbol de regresión.

- Ahora verificaremos la variable mas influyente dentro del modelo:

	%IncMSE	IncNodePurity
CompPrice	35.324343	230.55353
Income	7.923790	118.79213
Advertising	19.736816	155.03099
Population	-3.479681	63.11975
Price	70.613500	673.11982
ShelveLoc	68.147204	637.69500
Age	20.964215	228.90345
Education	4.705263	63.13124
Urban	-2.098091	11.66651
US	1.389570	11.15329

Con ayuda de la funcion importance(), se concluye que el precio (variable "Price"), es claramente la covariable más importante para predecir el comportamiento de las ventas ("Sales").

e) Bosque aleatorio (Random-Forest):

Ahora se aplicará bosque aleatorio (random forest) al conjunto de datos Carseats, utilizando el paquete randomForest.

- Como por defecto, randomForest() utiliza p/3 variables al construir un bosque aleatorio de árboles de regresión. En este caso utilizaremos mtry = 5.

```
set.seed(1)
bosque<-randomForest(Sales~.,data=Carseats,subset=train,mtry=5,importance=T)
yhat.rf<-predict(bosque,Carseats[-train,])
mean((yhat.rf-Carseats[-train,"Sales"])^2)
```

```
## [1] 2.60463
```

Podemos observar que el MSE de prueba asociado con el bosque aleatorio es de 2.605, un valor que si bien no mejora al hallado anteriormente con el método Bagging, sigue logrando reducir el MSE a casi la mitad del MSE de prueba hallado con el árbol de regresión.

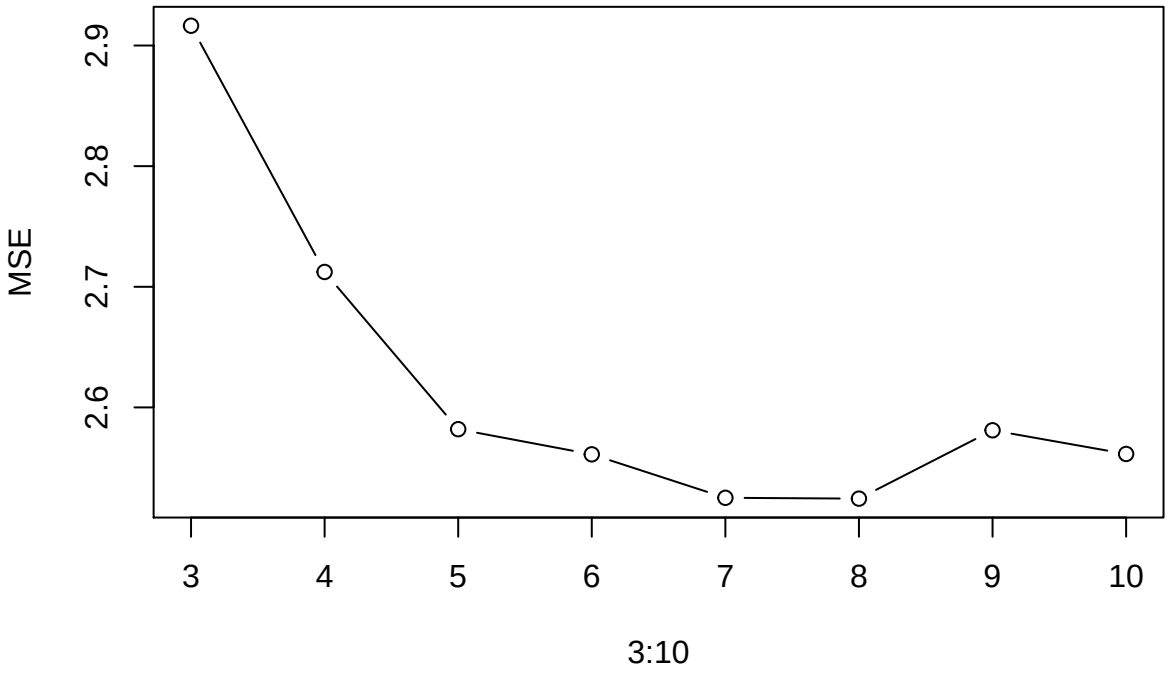
- Ahora verificaremos la variable mas influyente dentro del bosque aleatorio:

	%IncMSE	IncNodePurity
CompPrice	23.5396784	213.01972
Income	4.5652257	146.18592
Advertising	17.0723287	161.78851
Population	-0.5322612	99.34432
Price	56.8760700	622.42828
ShelveLoc	59.8410622	561.85844
Age	19.9678663	251.31101
Education	4.6628193	81.20597
Urban	-2.2973693	14.96249
US	4.4291018	19.99181

Con ayuda de la funcion importance(), se concluye que el precio y la calidad de ubicación (variables "Price" y "ShelveLoc"), son claramente las covariables más importantes para predecir el comportamiento de las ventas("Sales").

- Ahora describiremos el efecto de **m** (número de variables consideradas en cada subdivisión), iterando el modelo desde p/3 hasta el número total de predictores (covariables).

```
MSE<-c()
set.seed(1)
for(i in 3:10){
  bosqueit<-randomForest(Sales~.,data=Carseats,subset=train,mtry=i,importance=T)
  yhat.rfi<-predict(bosqueit,Carseats[-train,])
  MSE<-rbind(MSE,mean((yhat.rfi-Carseats[-train,"Sales"])^2))
}
plot(3:10,MSE,type="b")
```



Con la gráfica anterior, podemos ver que a medida que el valor de **m** cambia, el MSE de prueba decae, y concluimos que si consideramos solo 7 predictores para el entrenamiento podemos terminar con un MSE de prueba de 2.525, que es incluso menor al logrado con el método Bagging.

2. (MSVs - aplicado)En este ejercicio, se utilizará el enfoque de máquinas de soporte vectorial para predecir si un automóvil determinado posee un alto o bajo consumo de combustible basado en el conjunto de datos Auto (librería ILSR).

```
#librerias
library(e1071)
library(ISLR)
```

a). Cree una variable binaria que tome un 1 para automóviles con millaje por galón por encima de la mediana, y un 0 para automóviles con millaje por debajo de la mediana.

```
data(Auto)

m <- median(Auto$mpg)
y <- ifelse(Auto$mpg>m, 1,0)
y <- as.factor(y)
dat<- data.frame(Auto,y)
```

b). Ajuste un clasificador de soporte vectorial a los datos con varios valores del parámetro cost para predecir si un automóvil posee millaje altomo bajo. Informe los errores de validación cruzada asociados con diferentes valores de este parámetro. Comente sobre sus resultados.

```
#validacion cruzada para mirar el mejor smv con varios valores para cost
set.seed(32668)
vec <- tune(svm ,y~.,data=dat ,kernel ="linear",
              ranges =list(cost=c( 0.01, 0.1,0.5, 1,10,100,1000) ))
```

```
summary(vec)
```

```
##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.007692308
##
## - Detailed performance results:
##    cost       error dispersion
## 1 1e-02 0.073910256 0.04414558
## 2 1e-01 0.045897436 0.02906486
## 3 5e-01 0.012756410 0.01808165
## 4 1e+00 0.007692308 0.01238579
## 5 1e+01 0.017948718 0.02110955
## 6 1e+02 0.030705128 0.02357884
## 7 1e+03 0.030705128 0.02357884
```

Con cost=1 proporciona el menor error de validación cruzada, el cual fue 0.007692308.

c) Ahora repita **b)**, esta vez utilizando una máquina de soporte vectorial (svm) con una base de kernels radiales y polinomiales, con diferentes valores de los hiperparámetros cost, gamma o degree según el kernel y .comente sus resultados.

kernel radial

```
set.seed(23522)
vec <- tune(svm , y~., data=dat, kernel = "radial",
              ranges=list(cost=c(0.1,1,10,100,1000),
                          gamma=c(0.5,1,2,3,4) ))
```

```
summary(vec)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.04596154
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.07910256 0.04058164
## 2  1e+00   0.5 0.04602564 0.03782208
## 3  1e+01   0.5 0.04596154 0.02888973
## 4  1e+02   0.5 0.04596154 0.02888973
## 5  1e+03   0.5 0.04596154 0.02888973
## 6  1e-01   1.0 0.55365385 0.03913705
## 7  1e+00   1.0 0.06128205 0.04841637
## 8  1e+01   1.0 0.06128205 0.05134540
## 9  1e+02   1.0 0.06128205 0.05134540
## 10 1e+03   1.0 0.06128205 0.05134540
## 11 1e-01   2.0 0.55365385 0.03913705
## 12 1e+00   2.0 0.11250000 0.09473562
## 13 1e+01   2.0 0.10987179 0.08036018
## 14 1e+02   2.0 0.10987179 0.08036018
## 15 1e+03   2.0 0.10987179 0.08036018
```

```
## 16 1e-01   3.0 0.55365385 0.03913705
## 17 1e+00   3.0 0.39506410 0.15314896
## 18 1e+01   3.0 0.37211538 0.14680319
## 19 1e+02   3.0 0.37211538 0.14680319
## 20 1e+03   3.0 0.37211538 0.14680319
## 21 1e-01   4.0 0.55365385 0.03913705
## 22 1e+00   4.0 0.49474359 0.04938237
## 23 1e+01   4.0 0.48448718 0.05930309
## 24 1e+02   4.0 0.48448718 0.05930309
## 25 1e+03   4.0 0.48448718 0.05930309
```

Con varios valores para cost (10, 100,1000) y gamma=0.5 proporciona el menor error de validación cruzada, el cual fue 0.04596154.

kernel polinomial

```
set.seed(8527)
tune.out=tune(svm , y~., data=dat, kernel = "polynomial",
              ranges=list(cost=c(0.01,0.1,1),degree=c(2,3,4),
                          gamma=c(0.5,1,2,3) ))
```

```
summary(tune.out)
```

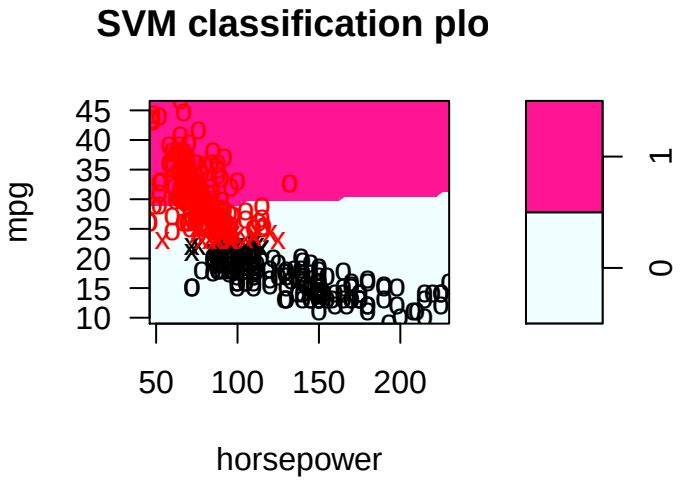
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree gamma
##   0.1      3     2
##
## - best performance: 0.04096154
##
## - Detailed performance results:
##    cost degree gamma      error dispersion
## 1  0.01      2   0.5 0.28314103 0.05292943
## 2  0.10      2   0.5 0.18102564 0.07060678
## 3  1.00      2   0.5 0.15544872 0.06389228
## 4  0.01      3   0.5 0.07935897 0.05480159
## 5  0.10      3   0.5 0.04615385 0.03783922
## 6  1.00      3   0.5 0.04102564 0.04554842
## 7  0.01      4   0.5 0.18384615 0.07164068
## 8  0.10      4   0.5 0.18121795 0.06907102
## 9  1.00      4   0.5 0.19128205 0.04503957
## 10 0.01      2   1.0 0.27012821 0.07219579
## 11 0.10      2   1.0 0.15294872 0.06467000
## 12 1.00      2   1.0 0.16070513 0.06150881
## 13 0.01      3   1.0 0.04615385 0.03783922
## 14 0.10      3   1.0 0.04358974 0.04689185
## 15 1.00      3   1.0 0.04352564 0.04021235
## 16 0.01      4   1.0 0.18115385 0.05338905
## 17 0.10      4   1.0 0.18871795 0.03406522
## 18 1.00      4   1.0 0.18352564 0.05818565
## 19 0.01      2   2.0 0.16320513 0.07248864
## 20 0.10      2   2.0 0.15544872 0.06156313
## 21 1.00      2   2.0 0.16583333 0.04384879
## 22 0.01      3   2.0 0.04358974 0.04689185
## 23 0.10      3   2.0 0.04096154 0.03669302
## 24 1.00      3   2.0 0.04352564 0.04021235
## 25 0.01      4   2.0 0.17858974 0.04191434
## 26 0.10      4   2.0 0.18352564 0.05818565
```

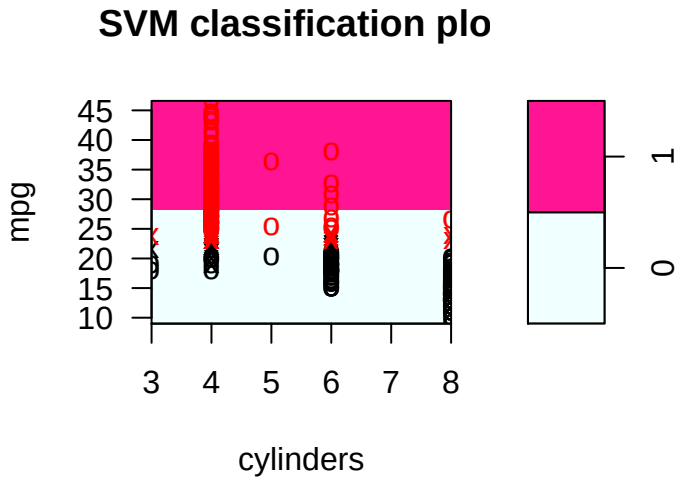
```
## 27 1.00      4   2.0 0.18352564 0.05818565
## 28 0.01      2   3.0 0.15038462 0.06513709
## 29 0.10      2   3.0 0.16070513 0.06150881
## 30 1.00      2   3.0 0.17608974 0.04908399
## 31 0.01      3   3.0 0.04352564 0.04369483
## 32 0.10      3   3.0 0.04352564 0.04021235
## 33 1.00      3   3.0 0.04352564 0.04021235
## 34 0.01      4   3.0 0.18352564 0.05818565
## 35 0.10      4   3.0 0.18352564 0.05818565
## 36 1.00      4   3.0 0.18352564 0.05818565
```

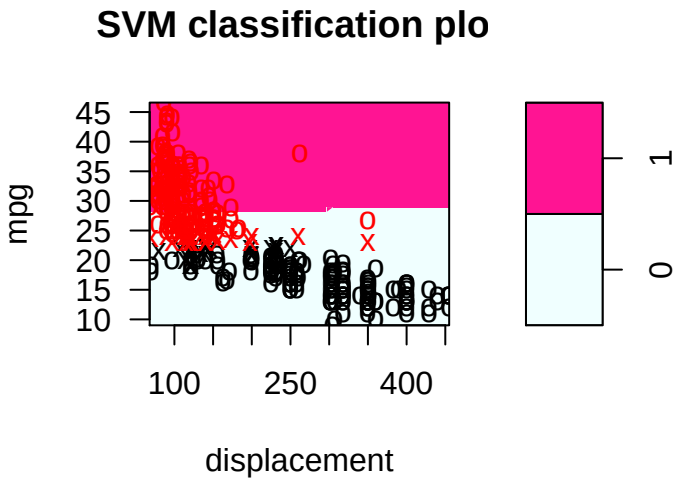
Con cost=0.1, gamma=2 y degree=3 proporciona el menor error de validación cruzada, el cual fue 0.04096154.

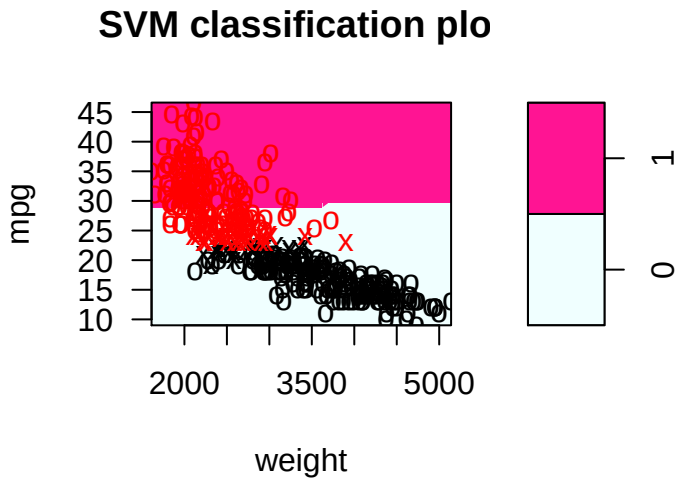
d). Realice algunos plots que sirvan de apoyo a sus afirmaciones en (b) y (c). Recomendación: En el lab, se utilizó la función plot() para objetos svm solo en casos con p = 2. Cuando p > 2, se puede utilizar la función plot() para crear gráficos que muestran pares de variables a la vez.

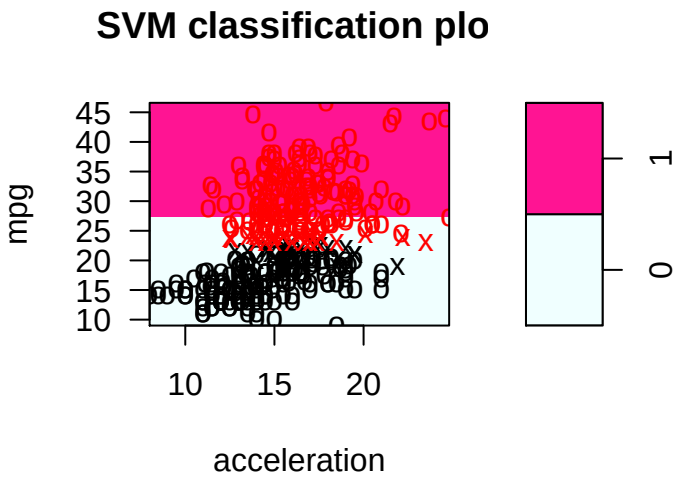
```
#modelo ajustado kernel linear
svm1 =svm(y~., data=dat, kernel = "linear",cost=1)
```

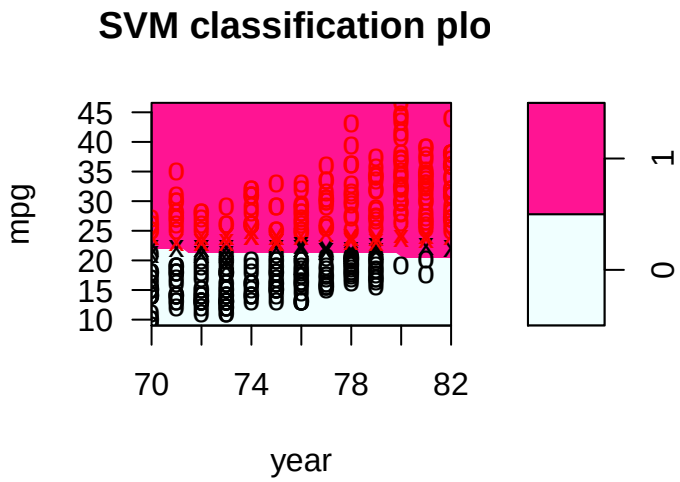


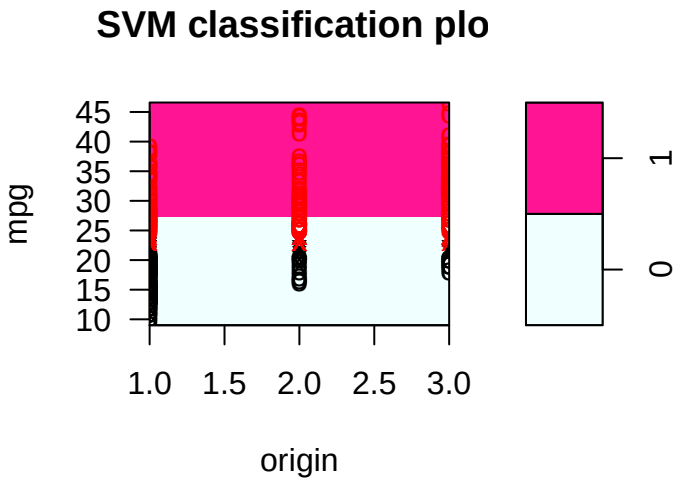






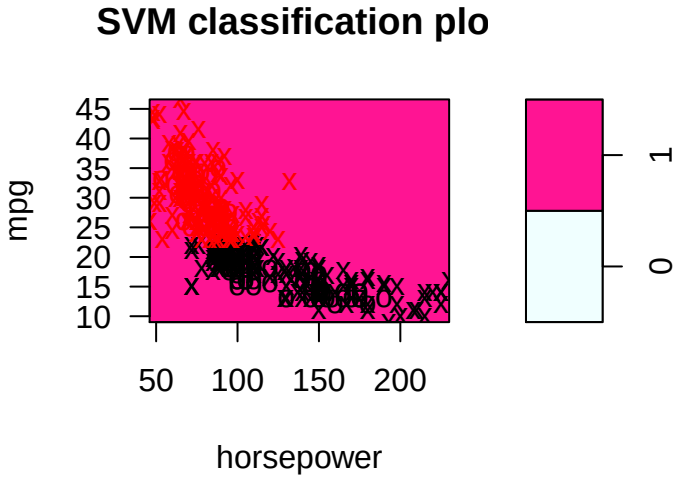


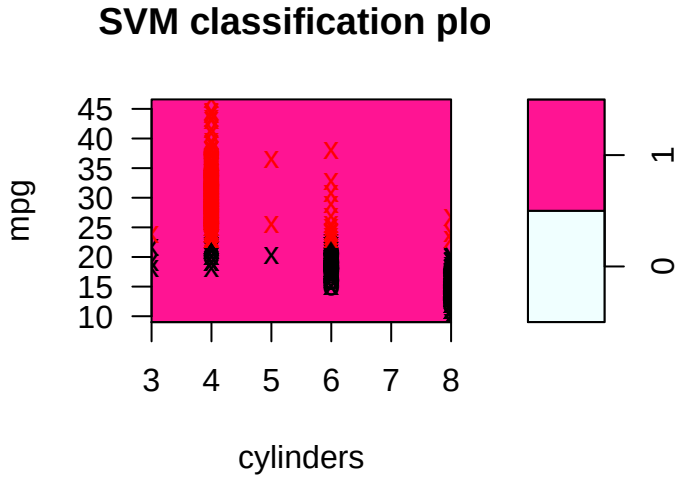


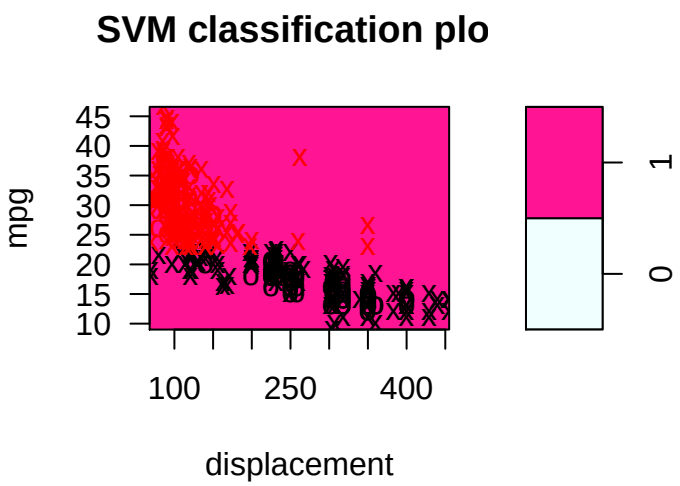


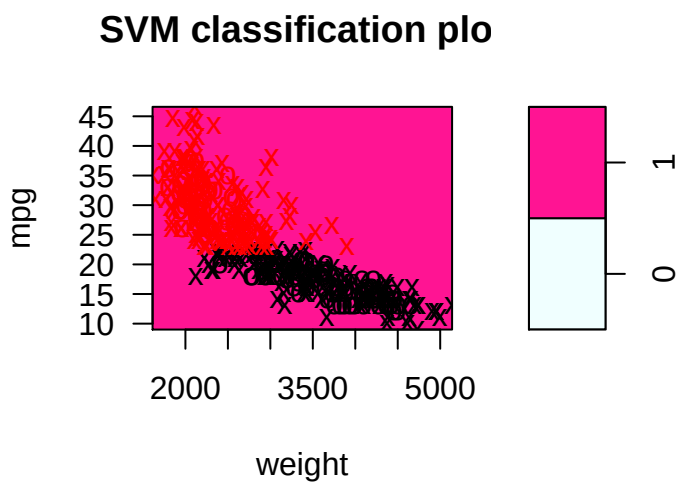
En los gráficos anteriores se observa como clasifica svm para las diferentes variables tomando de a dos, con el valor de cost que dio el error más bajo en la validación cruza.

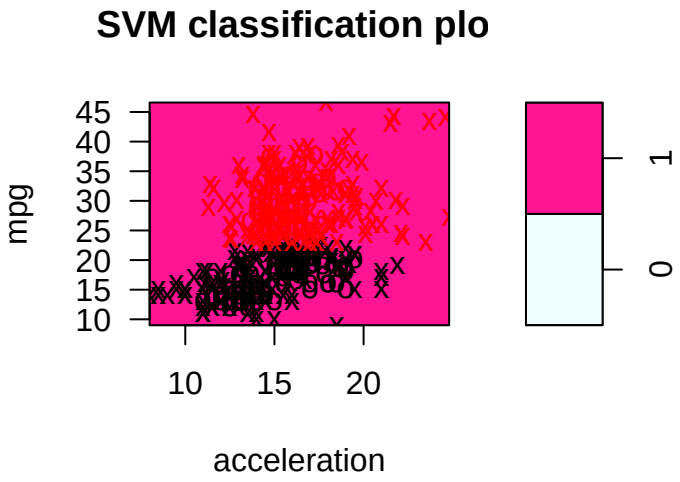
```
#modelo ajustado  kernel radial
svm2 =svm(y~., data=dat, kernel ="radial",cost=10,gamma=0.5)
```

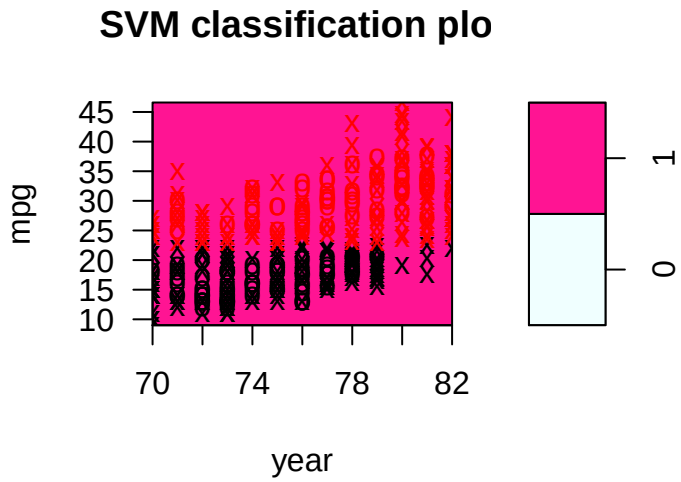


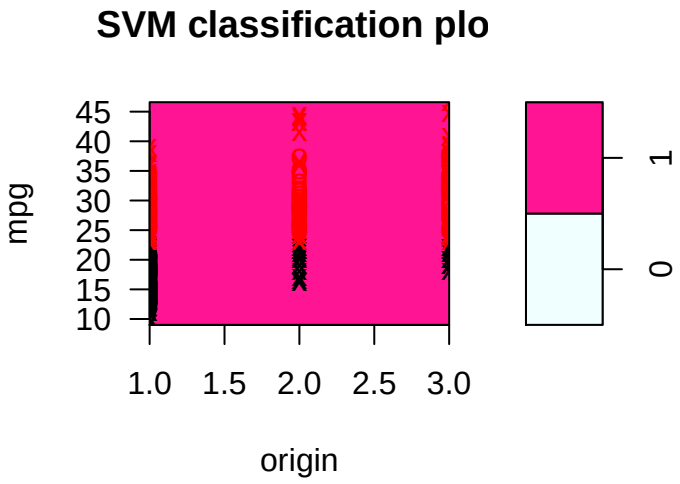








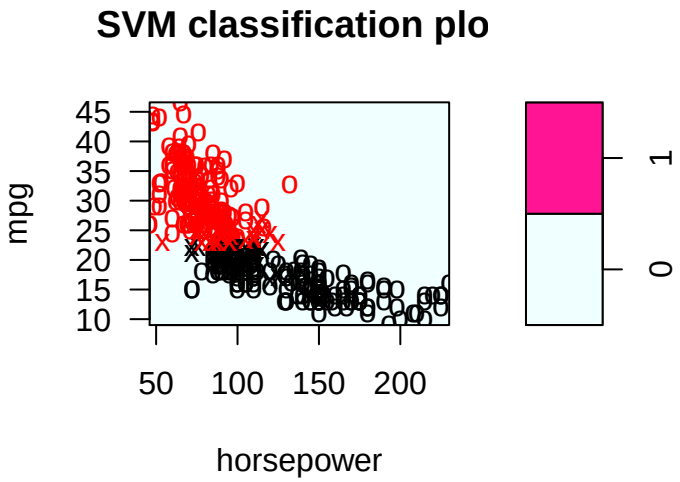


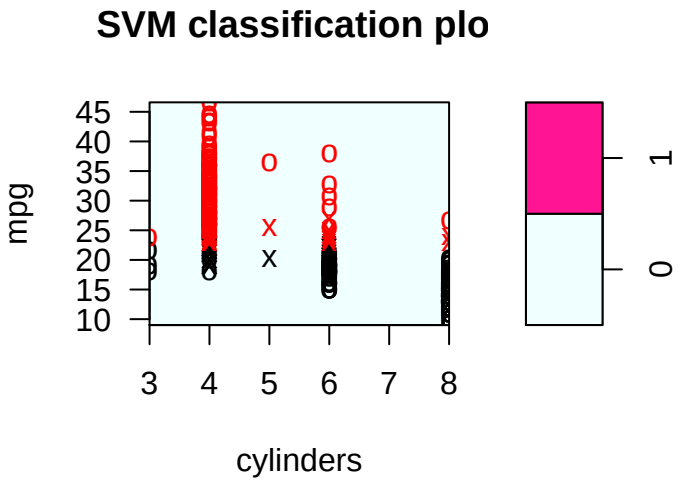


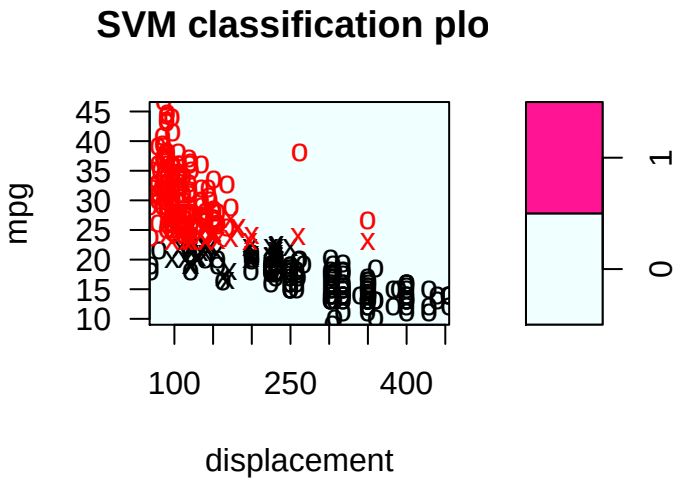
En los gráficos anteriores se observa como clasifica svm para las diferentes variables tomando de a dos, con el valor de cost=10 y gamma=0.05 que fue primero que se detectó con el error más bajo en la validación cruza.

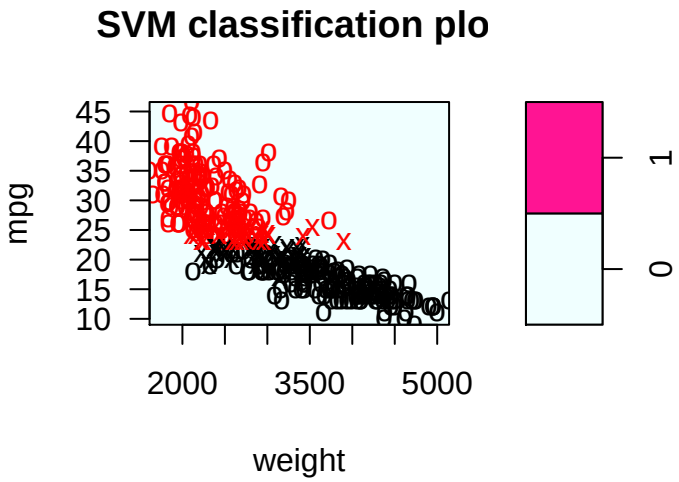
```
#modelo ajustado  kernel polinomial
svm3 =svm(y~., data=dat, kernel="polynomial",
```

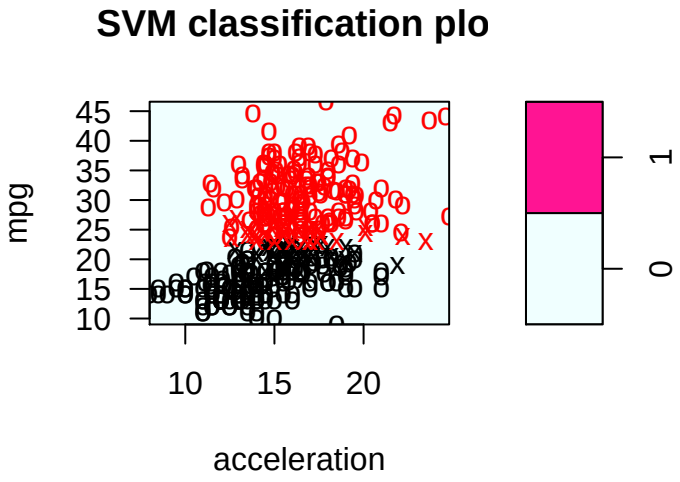
```
        cost=0.1,gamma=2,degree=3)
```

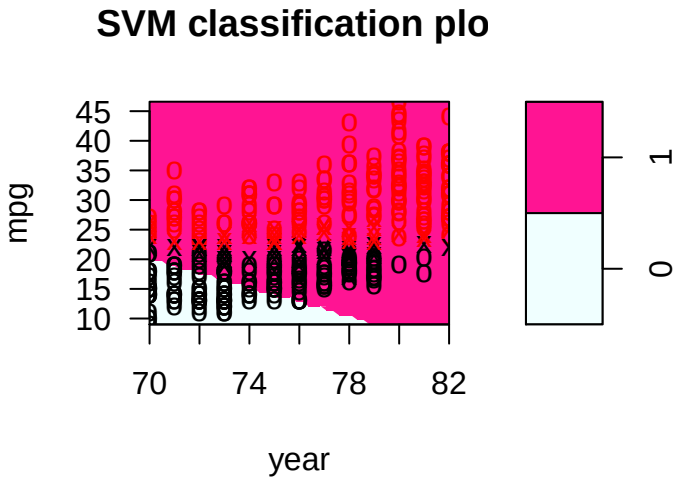


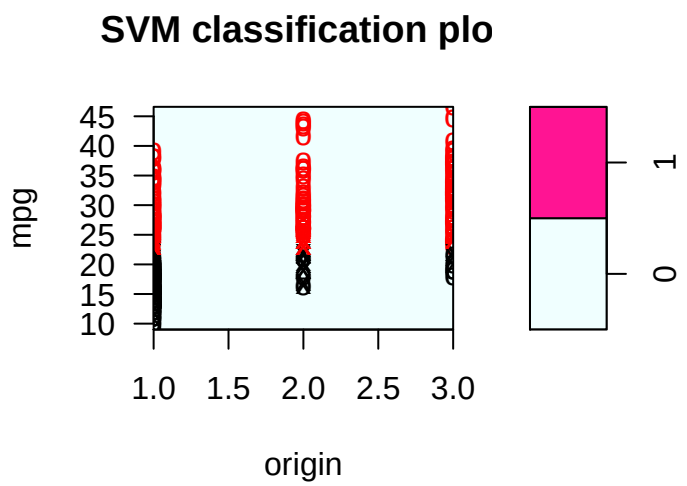












En los gráficos anteriores se observa como clasifica svm para las diferentes variables tomando de a dos, con el valor de cost=0.1, gamma=2 y degree=3 que fue uno de los que se detectó con el error más bajo en la validación cruza.

3. a. Datos

Para realizar este punto se utilizó la base de datos ISLR de donde se tomo la base de datos OJ, la cual se dividio en dos: conjunto de prueba y conjunto de entrenamiento.

```
library(dplyr)
library(ISLR)
library(e1071)

data <- ISLR::OJ

set.seed(1234)
subset <- sample((1:nrow(data)), 800)
train <- data[subset,]
test <- data[-subset,]
```

b. Clasificador con kernel lineal

Con la ayudad de la función svm de la libreria e1071, se ajustó un clasificador de soporte vectorial con kernel lineal y con un valor para el parámetro cost de 0.1. Este modelo arroja el siguiente summary:

```
svm_linear <- svm(Purchase~., data=train , kernel = "linear", cost = 0.1,
                  scale = FALSE )
summary(svm_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.1,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  438
##
##  ( 219 219 )
##
```

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

De la anterior salida vemos que se obtuvieron 438 vectores de soporte, donde 219 pertenecen a una clase y 219 a la otra. Es decir, para cada clase se obtuvo el mismo número de vectores de soporte.

c. Tasas de error

Para observar el error en la clasificación, se predicen las estiquetas de clase tanto para el conjunto de entrenamiento como para el conjunto de prueba.

Conjunto de entrenamiento

```
train_linear <- predict(svm_linear, train)
table(prediccion = train_linear, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 426  78
##         MM  57 239
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 16.875
```

De la anterior tabla, vemos que 665 observaciones han sido clasificadas de manera correcta y 135 fueron mal clasificadas, por lo que el 16.88% de estas observaciones fueron mal clasificadas.

Conjunto de prueba

```
test_linear <- predict(svm_linear, test)
table(prediccion = test_linear, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 151  24
##         MM  19  76
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15.92593
```

De esta tabla observamos que 227 observaciones fueron bien clasificadas y 43 fueron mal clasificadas, por lo que el 15.93% de las observaciones fueron mal clasificadas.

d. Valor óptimo para el parámetro cost.

Con la ayuda de la función cost se obtiene el valor óptimo para el parámetro cost, el cual varia en el rango de valores de 0.01 a 10.

```
tune_linear <- tune(svm, Purchase~., data=train ,kernel ="linear",
                    ranges = list(cost=c(0.01, 0.1, 1,5,10)))
summary(tune_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
```

```
##
## - best performance: 0.17
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17125 0.03866254
## 2  0.10 0.17000 0.04297932
## 3  1.00 0.17250 0.04401704
## 4  5.00 0.17500 0.04124790
## 5 10.00 0.17375 0.04185375
```

Del summary vemos que el menor error es de 0.17000, el cual se obtiene con un valor de cost de 0.10. Ahora se procede a obtener un mejor modelo con este valor de cost.

```
best_linear <- tune_linear$best.model
summary(best_linear)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  344
##
##  ( 173 171 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Se aprecia que con un valor de cost de 0.10, se obtuvieron 344 vectores de soporte, 173 en una clase y 171 en la otra. Note, que con este nuevo ajuste se tiene 94 vectores de soporte menos que con el ajuste incial.

e. Tasas de error del modelo con el valor óptimo de cost

Se predicen las etiquetas de clase tanto para el conjunto de prueba como el de entrenamiento, para el modelo ajustado con el valor ótimo obtenido en el anterior literal.

Conjunto de entrenamiento

```
pred_best_linear <- predict(best_linear, train)
table(prediccion = pred_best_linear, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 424  73
##         MM  59 244
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 16.5
```

Se observa que 668 observaciones fueron clasificadas de forma correcta y 132 fueron mal clasificadas. Por lo que el 16.5% de las observaciones fueron mal clasificadas.

Conjunto de prueba

```
pred_best_linear <- predict(best_linear, test)
table(prediccion = pred_best_linear, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 148  22
##         MM  22  78
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 16.2963
```

De esta tabla se ve que 226 de las observaciones fueron clasificadas de manera correcta y 44 fueron mal clasificadas, Por lo que el 16.30% de loa observaciones fueron mal clasificadas.

Note que el valor óptimo que se encontro para el parámetro cost es de 0.1, el cual es el mismo valor del ajuste inicial, sin embargo cuando se hace un nuevo ajuste y se calculan las tasas de error se aprecian "pequeñas" diferencias en los porcentajes obtenidos.

f. Clasificador con kernel radial

Se ajusta un nuevo clasificador de soporte vectorial, pero esta vez con kernel radial, con un valor de 0.1 para el parámetro cost y con el valor por defecto para γ.

```
svm_radial <- svm(Purchase~., data = train, kernel = "radial", cost =0.1)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "radial", cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
## Number of Support Vectors:  553
##
##  ( 278 275 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

De este summary se obtiene 553 vectores de soporte, 278 en una clase y 275 en la otra clase.

A continuación se predicen las etiquetas de clase tanto para el conjunto de entrenamiento como el de prueba.

Conjunto de entrenamiento

```
train_radial <- predict(svm_radial, train)
table(prediccion = train_radial, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 429  87
##         MM  54 230
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 16.875
```

Observamos que 659 de las observacones fueron clasificadas de manera correcta y 141 fueron clasificadas erroneamente. Por lo que el 16.88% de las observaciones estan mal clasificadas.

Conjunto de prueba

```
test_radial <- predict(svm_radial, test)
table(prediccion = test_radial, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 153  25
##         MM  17  75
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15.92593
```

Vemos quue 228 de las observaciones fueron clasificadas de manera correcta y 42 fueron mal clasificadas. Por lo que el 15.93% de las observaciones fueron mal clasificadas.

Ahora, con la función tune() se procede hallar un valor óptimo para el parámetro cost.

```
tune_radial <- tune(svm, Purchase~., data=train ,kernel ="radial",
                    ranges = list(cost=c(0.01, 0.1, 1,5,10)))
summary(tune_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17125
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39625 0.04825065
## 2  0.10 0.19000 0.05163978
## 3  1.00 0.17125 0.03821086
## 4  5.00 0.17125 0.03175973
## 5 10.00 0.18500 0.03374743
```

De este summary podemos apreciar que el menor error es 0.17125 el cual se obtiene con un valor de cost de 1 y 5. Ahora, se obtiene el mejor ajuste con las siguientes lineas de código:

```
best_radial <- tune_radial$best.model
summary(best_radial)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  375
```

```
##
##  ( 188 187 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Observamos que el valor de cost elegido es de 1. Además, con este ajuste se obtuco 375 vectores de soporte, 188 en una clase y 187 en la otra.

Ahora, predecimos las etiquetas de clase para cada conjunto de datos.

Conjunto de entrenamiento

```
pred_best_radial <- predict(best_radial, train)
table(prediccion = pred_best_radial, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 442  79
##         MM  41 238
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15
```

Se observa que 680 de las observaciones fueron clasificadas de manera correcta y 120 fueron mal clasificadas. Por lo que el 15% de las observacones fueron clasificadas erroneamente.

Conjunto de prueba

```
pred_best_radial <- predict(best_radial, test)
table(prediccion = pred_best_radial, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 154  26
##         MM  16  74
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15.55556
```

Se observa que 228 de las observaciones fueron clasificadas de manera correcta y 42 fueron mal clasificadas. Por lo que el 15.56% de las observacones fueron clasificadas de forma incorrecta.

g. Clasificador con kernel polinomial

A continuación se ajusta una maquina de soporte vectorial con kernel polinomial, un valor para el parámetro cost de 0.1 y con un valor de 2 para el parámetro degree.

```
svm_poli <- svm(Purchase~., data = train, kernel = "polynomial",
                degree = 2, cost = 0.1)
summary(svm_poli)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2, cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  polynomial
##        cost:  0.1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  601
##
##  ( 303 298 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Vemos del anterior summary que se obtuvieron 601 vectores de soporte, 303 en una clase y 298 en la otra.

Ahora, predecimos las etiquetas de clase para ambos conjuntos de datos.

Conjunto de entrenamiento

```
train_poli <- predict(svm_poli, train)
table(prediccion = train_poli, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 462 235
##         MM  21  82
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 32
```

Vemos que 544 de las observaciones fueron clasificadas de forma correcta y 256 fueron mal clasificadas. Por lo que el 32% de las observaciones fueron mal clasificadas.

Conjunto de prueba

```
test_poli <- predict(svm_poli, test)
table(prediccion = test_poli, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 163  65
##         MM   7  35
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 26.66667
```

Observe que 198 de las observaciones fueron clasificadas de forma correcta y 72 fueron mal clasificadas. Por lo que el 26.67% de las observaciones fueron clasificadas de forma incorrecta.

Ahora, usamos la función tune() para obtener un valor óptimo para el parámetro cost.

```
tune_poli <- tune(svm, Purchase~., data=train ,kernel ="polynomial",
                  degree = 2,ranges = list(cost=c(0.01, 0.1, 1,5,10)))
summary(tune_poli)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
```

```
##  cost
##     5
##
## - best performance: 0.18625
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39250 0.04866267
## 2  0.10 0.33375 0.05104804
## 3  1.00 0.20500 0.04609772
## 4  5.00 0.18625 0.03304563
## 5 10.00 0.18750 0.02204793
```

Observamos que el menor error es de 0.18625, el cual se obtiene con un valor de 5 para el parámetro cost. Ahora, obtenemos un mejor modelo con este valor de cost.

```
best_poli <- tune_poli$best.model
summary(best_poli)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10)), kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  373
##
##  ( 190 183 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Observe que en este caso se obtienen 373 vectores de soporte, 190 en una clase y 183 en la otra. Además, note que se obtuvieron 228 vectores de soporte menos que con el ajuste incial con un valor de cost de 0.1.

Ahora procedemos a predecir la etiquetas de clase para los conjuntos de entrenamiento y de prueba.

```
pred_best_poli <- predict(best_poli, train)
table(prediccion = pred_best_poli, real = train$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 446  86
##         MM  37 231
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15.375
```

Observamos que 677 de las observaciones fueron clasificadas de forma correcta y 123 fueron clasificadas de forma incorrecta. Por lo que el 15.38% de las observaciones fueron mal clasificadas.

Conjunto de prueba

```
pred_best_poli <- predict(best_poli, test)
table(prediccion = pred_best_poli, real = test$Purchase)
```

```
##           real
## prediccion  CH  MM
##         CH 155  28
##         MM  15  72
```

```
## [1] "Porcentaje mal clasificado:"
```

```
## [1] 15.92593
```

Se aprecia que 227 de las observaciones fueron clasificadas de forma correcta y 43 fueron clasificadas de forma incorrecta. Por lo que el 15.93% de las observaciones fueron mal clasificadas.

h. Comparando resultados

De los resultados obtenidos en los literales anteriores se observa que con un ajuste con kernel linel, con un valor de cost de 0.1, se obtienen 344 vectores de soporte, un error de entrenamiento de 16.5% y un error de prueba de 16.30%. Con un ajuste con kerner radial, con un valor de cost de 1, se obtienen 375 vectores de soporte, un error de entrenamiento de 15% y un error de prueba de 15.56% y con un ajuste con kernel polinomial, con un valor de cost de 0.1, se obtienen 373 vectores de soporte, un error de entrenamiento de 15.38% y un error de prueba de 15.93%. Por lo tanto, se tiene que en general si se trabaja con el conjunto de entrenamiento o con el conjunto de prueba el mejor clasificador de soporte vectorial es el ajustado con kernel radial, pues es con este con el que mejor resultados se obtienen con estos datos.

4.

```
#librerias
library(FactoMineR)#PCA
library(factoextra)
library(gridExtra)#par
library(kableExtra)#tablas
library(dplyr)
```

[PCA, K-medias] En este ejercicio Ud va a generar un conjunto simulado de datos y entonces aplicará PCA y agrupamiento por k-medias sobre dichos datos.

a)

Genere un conjunto de datos simulados con 20 observaciones en cada una de tres clases (es decir, 60 observaciones en total) y 50 variables. Sugerencia: hay una serie de funciones en R que puede utilizar para generar datos. Un ejemplo es la función rnorm(); runif() es otra opción. Asegúrese de agregar un cambio en la media en las observaciones de cada clase a fin de obtener tres clases distintas.

Solución

Primero se fija una semilla que permita la reproductibilidad de los datos obtenidos y luego con ayuda de la función rnorm() y runif() se simula una base de datos con 50 variables, 60 observaciones y 3 clases de población:

```
set.seed(123)
df<- data.frame(matrix(nrow=60,ncol = 50))
for(i in 1:50){
  a=rnorm(n = 20,52,3)
  b=rnorm(n = 20,72,5)
  c=runif(n = 20,min = 30,max = 55)
  df[,i] <- c(a,b,c)
}
df$clase <- c(rep("1",20),rep("2",20),rep("3",20))
```

A continuación, se puede apreciar la estuctura general, con las primeras y últimas filas y también columnas:

	X1	X2	X-	X50	clase
1	50.3185730603434	52.7599555419843	.....	53.2190994144543	1
2	51.3094675315502	51.9143597339539	.....	57.1425955773143	1
3	56.6761249424474	51.871388628126	.....	51.8188403376371	1
4	.....	.....	.....	.....	.....
58	32.339874667814	46.4807581051718	.....	35.7901265332475	3
59	41.669476039242	33.8086654210929	.....	48.4854441497009	3
60	42.7876364975236	44.3216764554381	.....	30.2025894296821	3

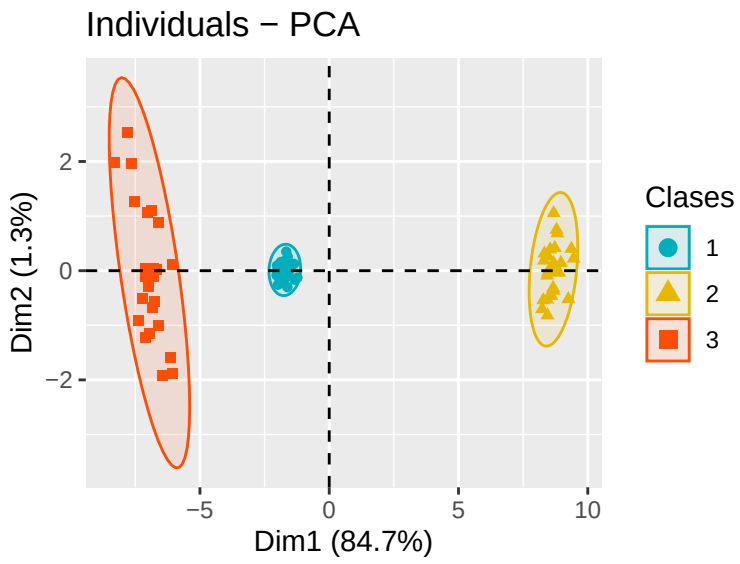
b)

Realice PCA en las 60 observaciones y grafique las observaciones en términos de las 2 primeras variables principales Z1 y Z2. Use un color diferente para indicar las observaciones en cada una de las tres clases. Si las tres clases aparecen separados en esta gráfica, solo entonces continúe con la parte (c). Si no, vuelva al inciso a) y modifique la simulación para que haya una mayor separación entre las tres clases. No continúe con la parte (c) hasta que las tres clases muestren al menos algún grado de separación en los dos primeros vectores de scores de componentes principales.

Solución

A continuación se realiza el ajuste de componentes principales (PCA), el cual intenta reducir la dimensionalidad de la base de datos, y luego se ilustran las observaciones en las primeras 2 componentes principales, la cual logra segmentar correctamente las clases. La primera componente retiene el 84.7% de la varianza total de los datos originales, mientras la segunda el 1.3%

```
#Ajuste
res.pca <- PCA(df[,-51],graph =F)
#Gráfico
fviz_pca_ind(res.pca,
             geom.ind = "point", #mostrar puntos
             col.ind = df$clase, #clases
             palette = c("#00AFBB", "#E7B800", "#FC4E07"),
             addEllipses = TRUE, #elipses
             legend.title = "Clases"
             )+ theme_gray()
```



c)

Desarrolle agrupación de K-medias de las observaciones con K = 3. ¿Qué tan bien funcionan los clústeres que obtuvo con el algoritmo de K-medias comparado con las verdaderas etiquetas de clase?

Sugerencia: puede usar la función table() en R para comparar las verdaderas etiquetas de clase con las etiquetas de clase obtenidas por agrupamiento. Tener cuidado cómo se interpretan los resultados: el agrupamiento de K-medias

numera los grupos arbitrariamente, por lo que no puede simplemente comprobar si las verdaderas etiquetas de clase y las etiquetas de agrupación son las mismas.

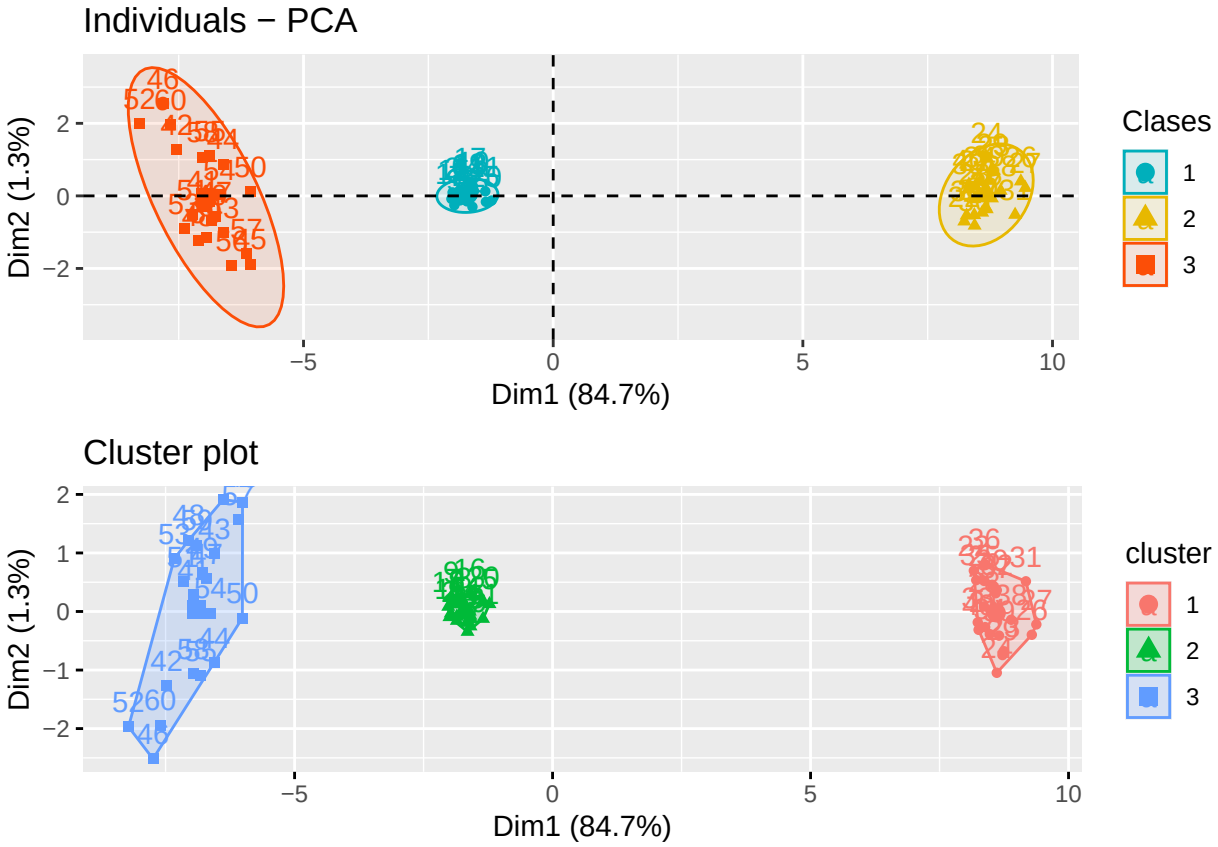
Solucón

Se realiza a continuación un análisis visual para saber como se están comportando las etiquetas que resultan de la función kmeans():

```
set.seed(123)
km.out =kmeans (df[,-51],3, nstart =20)
```

```
grid.arrange(
            fviz_pca_ind(res.pca,
             geom.ind = c("point","text"), #mostrar puntos
             col.ind = df$clase, #clases
             palette = c("#00AFBB", "#E7B800", "#FC4E07"),
             addEllipses = TRUE, #elipses
             legend.title = "Clases")+ theme_gray(),

            fviz_cluster(km.out, data = df[,-51]))
```



Así entonces, como el agrupamiento de K-medias numera las clases arbitrariamente, se verifican las clases a las que pertenecen los individuos y se puede notar que las etiquetas 1 y 2 están intercaladas. Luego, se realiza el siguiente procedimiento para intercarbiar dichos valores:

```
km2 <- km.out$cluster
for(i in 1:length(km2)){
  if(km2[i]==1){
    km2[i]<-2
  }else if(km2[i]==2){
    km2[i]<- 1
  }else{
    next
  }}; km2
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

De esta manera, logramos obtener las siguiente matriz de confusión, donde todas las observaciones han sido clasificadas correctamente. Es decir, que la tasa de error fue cero, por lo cual se conlcuye que logoritmo de K-medias comparado con las verdaderas etiquetas funciona correctamente.

```
table(km2,df$clase)
```

```
##
## km2  1  2  3
##   1 20  0  0
##   2  0 20  0
##   3  0  0 20
```

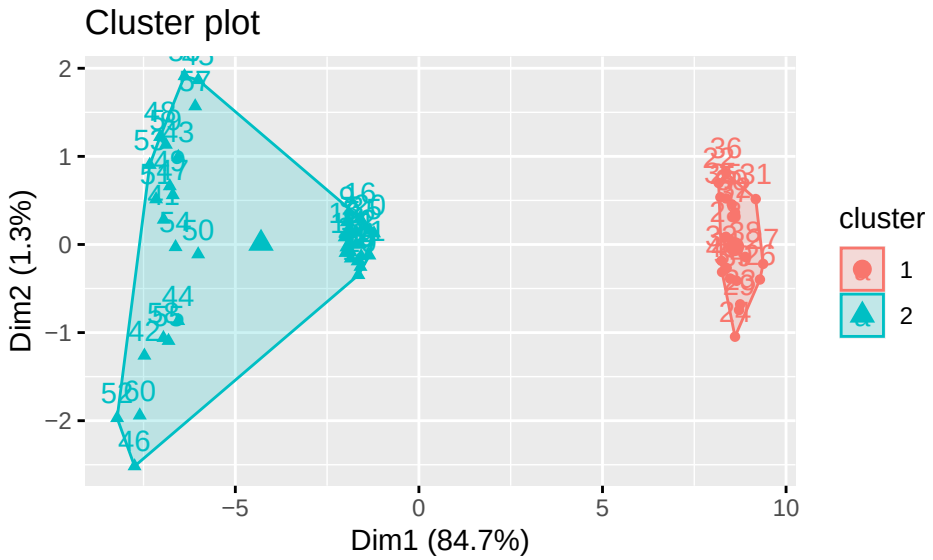
d)

Realice agrupamiento de K-medias con K = 2. Describa sus resultados.

Solución

En el siguiente gráfico podemos observar que realizando el agrupamiento de K-medias con K = 2, el algoritmo clasifica los datos en dos clases, para ello, en este caso unieron las 2 poblaciones más cercanas en una sola (cluster 2, azul), las cuales corresponden a la población original 1 con distribución uniforme y a la población original 3 con distribución normal.

```
set.seed(123) #semilla
km.out =kmeans (df[,-51],2, nstart =20)
fviz_cluster(km.out, data = df[,-51])
```



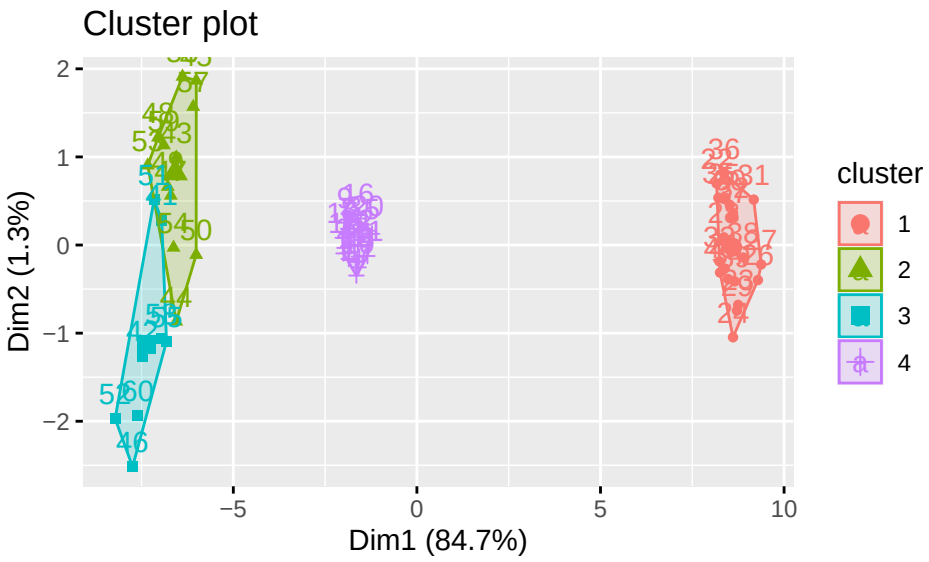
e)

Ahora realice agrupamiento de K-medias con K = 4 y describa sus resultados.

Solución

En el siguiente gráfico podemos observar que realizando el agrupamiento de K-medias con K = 4, el algoritmo clasifica los datos en cuatro clases, para ello, en este caso separó en dos la población con mayor dispersión (cluster 2 y 3) que corresponden el las verdaderas etiquetas a las poblaciones 1 y 3 respectivamente.

```
set.seed(123) #semilla
km.out =kmeans(df[,-51],4, nstart =20)
fviz_cluster(km.out, data = df[,-51])
```



f)

Ahora realice agrupamiento de K-medias con K = 3 en los dos primeros vectores de scores de componentes principales, en lugar de los datos en las variables originales. Es decir, realice la agrupación de K-medias en la matriz de 60 O 2, cuya primera columna es la coordenada zi1 en la primera componente principal Z1 y la segunda columna es la coordenada zi2 en la segunda componente principal Z2. Comente los resultados.

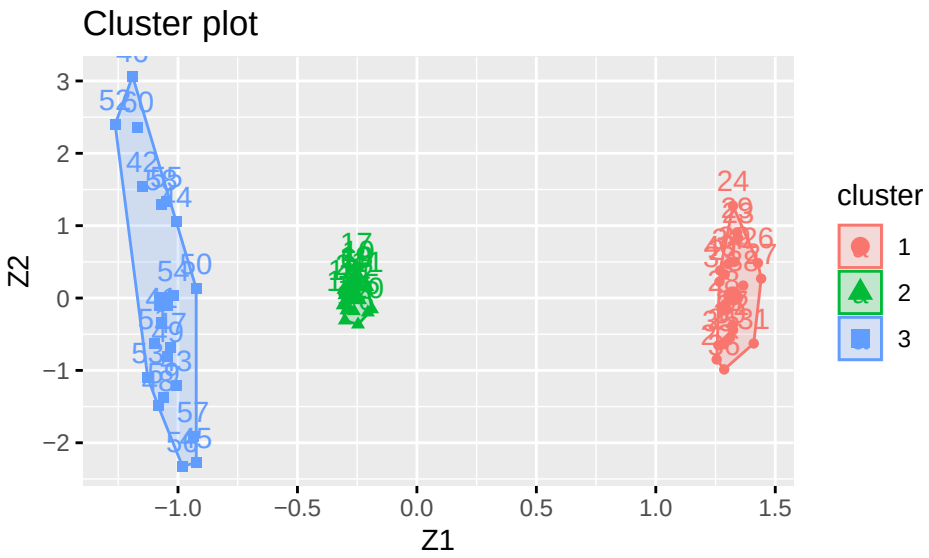
Solución

Primero se obtienen las coordenadas de las dos primeras componentes principales, de la siguiente manera:

```
Z1 <- res.pca$ind$coord[,1]
Z2 <- res.pca$ind$coord[,2]
Z<- data.frame(Z1,Z2)
```

Luego, se realiza la agrupación de K-medias con K = 3 con los datos de Z

```
set.seed(123)
km.out =kmeans(Z,3, nstart =20)
fviz_cluster(km.out, data = Z)
```



Como se puede observar, realizar agrupación de K medias con las coordenadas obtenidas en las dos primeras componenetes principales se logra también una correcta clasificación de los datos originales. Cabe mencionar que tener conocimiento acerca del número de clases, lo cual favorece los resultados del análisis y que se manifestó un cambio en escala con

respercto a la original.

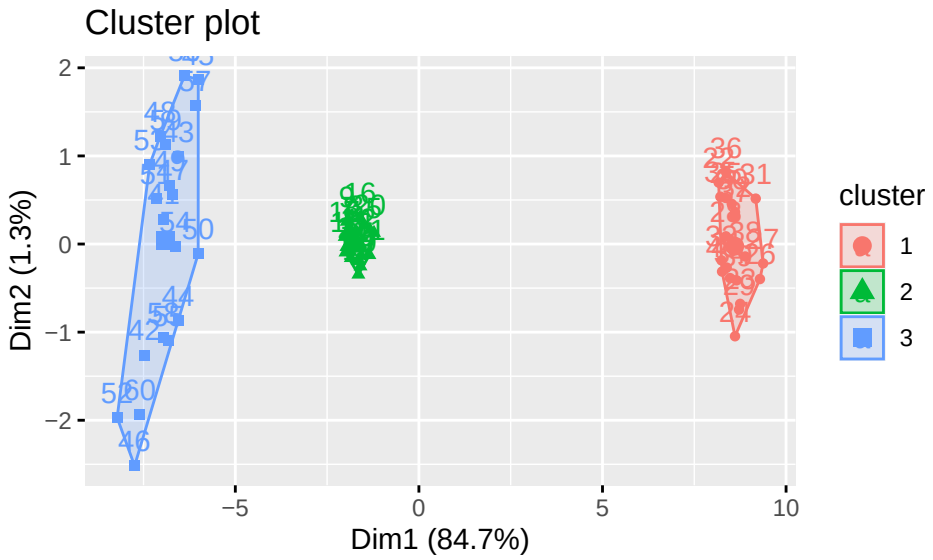
g)

Con la función scale(), realice agrupamiento de K-medias con K = 3 en los datos después de escalar cada variable para tener una desviación estándar de uno. ¿Cómo se comparan estos resultados con los obtenidos? en (b)? Explique.

Solución

```
set.seed(123)
sd.data=scale(df[,-51])
km.out =kmeans (sd.data,3, nstart =20)
```

```
fviz_cluster(km.out, data = sd.data)
```



En este caso, el agrupamiento de K-medias de los datos escalados, es decir con desviación estándar de uno, también logra clasificar correctamente las tres diferentes poblaciones. Es decir, con respecto a (b) donde se hizo uso de análisis de componentes principales, ambos logran clasificar correctamente los individuos de las tres diferentes distribuciones.

5. Considere el conjunto de datos **USArrests**. En este ejercicio se agruparán los estados en **USArrests** con agrupamiento jerárquico

```
#Librerias
library(ISLR)
library(factoextra)
```

```
data("USArrests")
states =row.names(USArrests )
head(USArrests)
```

```
##            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
```

Los datos de USArests contienen estadísticas (tasas), en arrestos por cada 100000 residentes por asalto, asesinato y violación en cada uno de los 50 estados de EE..UU, en 1973. También se da el porcentaje númerico de la población que vive en áreas urbanas.

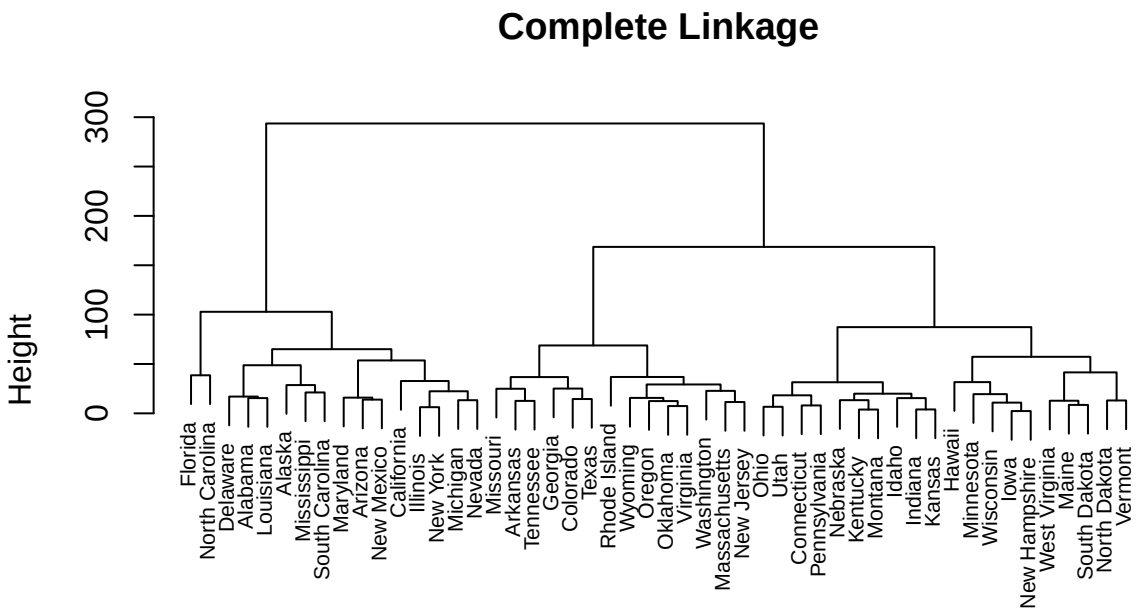
a). Utilice agrupación jerárquica con enlace completo y distancia euclidiana para agrupar los estados.

```
hc.complete =hclust(dist(USArrests), method ="complete")
hc.complete
```

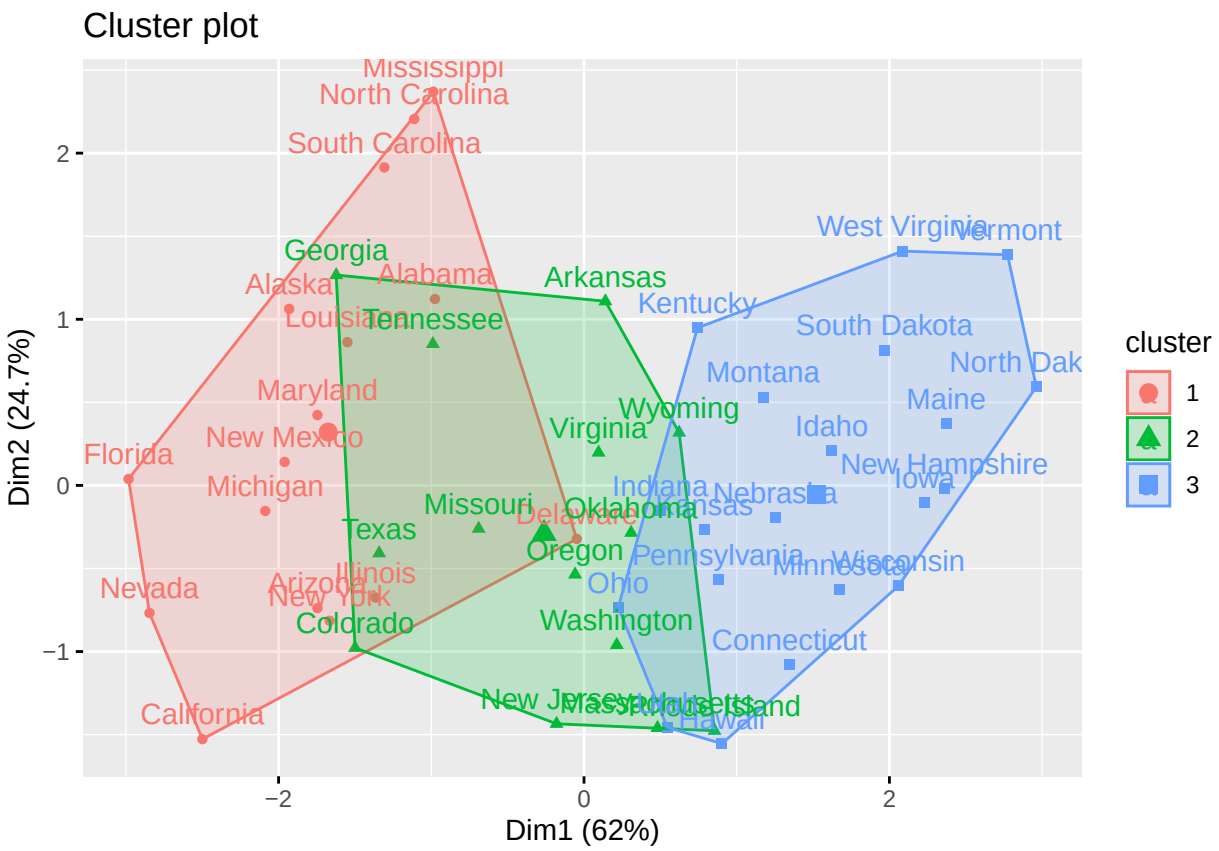
```
##
## Call:
## hclust(d = dist(USArrests), method = "complete")
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 50
```

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

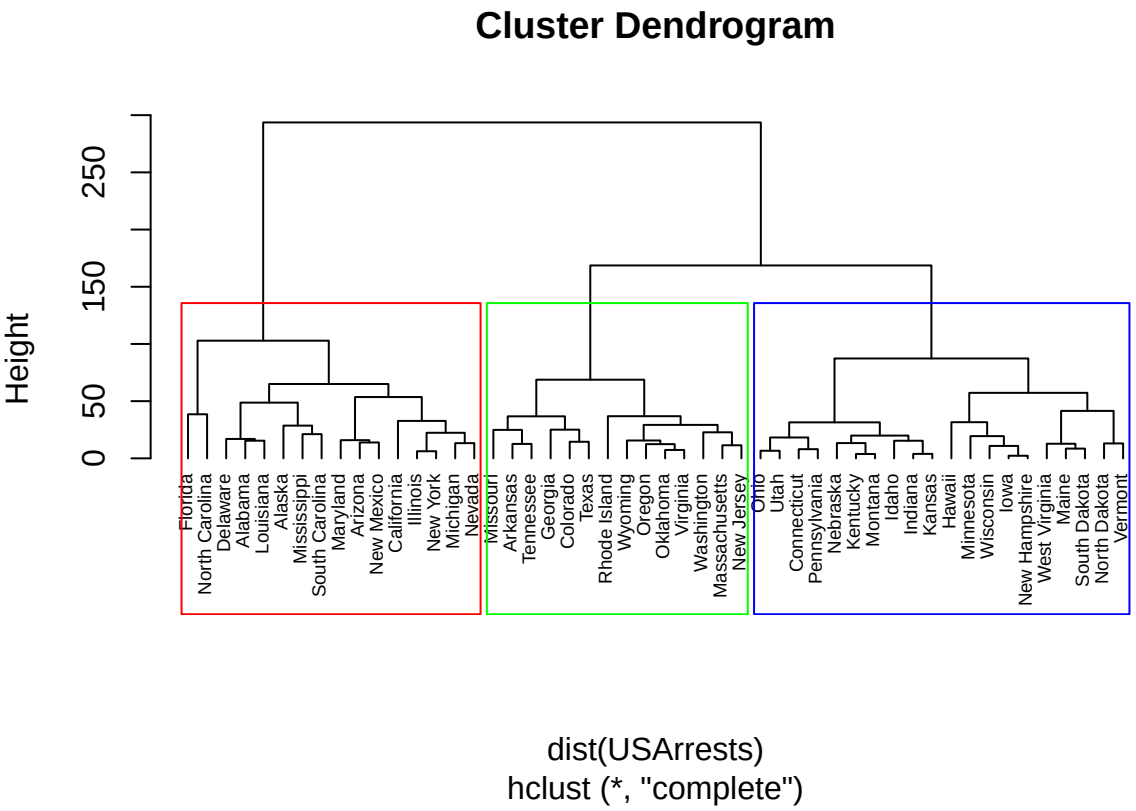
```
plot(hc.complete ,main =" Complete Linkage ", xlab="", sub ="",
cex =0.7)
```




```
clust <- cutree(hc.complete, k = 3)
fviz_cluster(list(data = USArrests, cluster = clust))
```




```
plot(hc.complete, hang = -1, cex = 0.6)
rect.hclust(hc.complete, k = 3, border = c("red","green","blue"))
```



Observando el anterior dendograma, se escogieron tres cluster como resultado, cada cluster le pertenecen los siguientes estados.

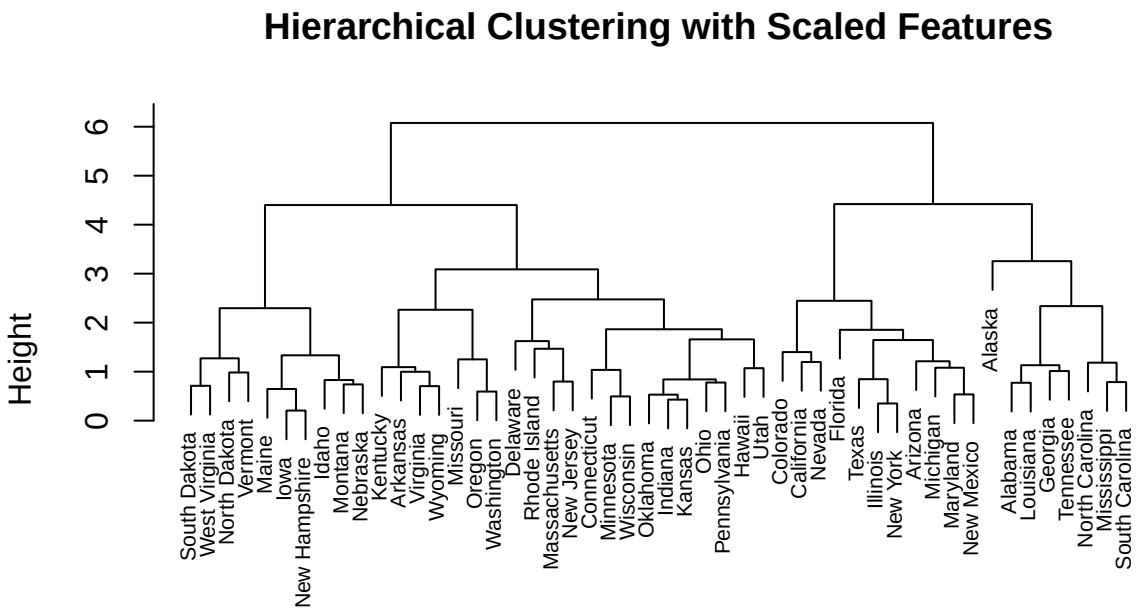
Cluster 1 (Rojo) Florida, Carolina del Norte, Delaware, Alabama, Luisiana, Alaska, Misisipi, Carolina del Sur, Maryland, Arizona, Nuevo México, California, Illinois, Nueva York, Michigan y Nevada.

Cluster 2 (Verde) Misuri, Arkansas, Tennessee, Georgia, Colorado, Texas, Rhode Island, Wyoming, Oregon, Oklahoma, Virginia, Washington, Massachusetts y Nueva Jersey.

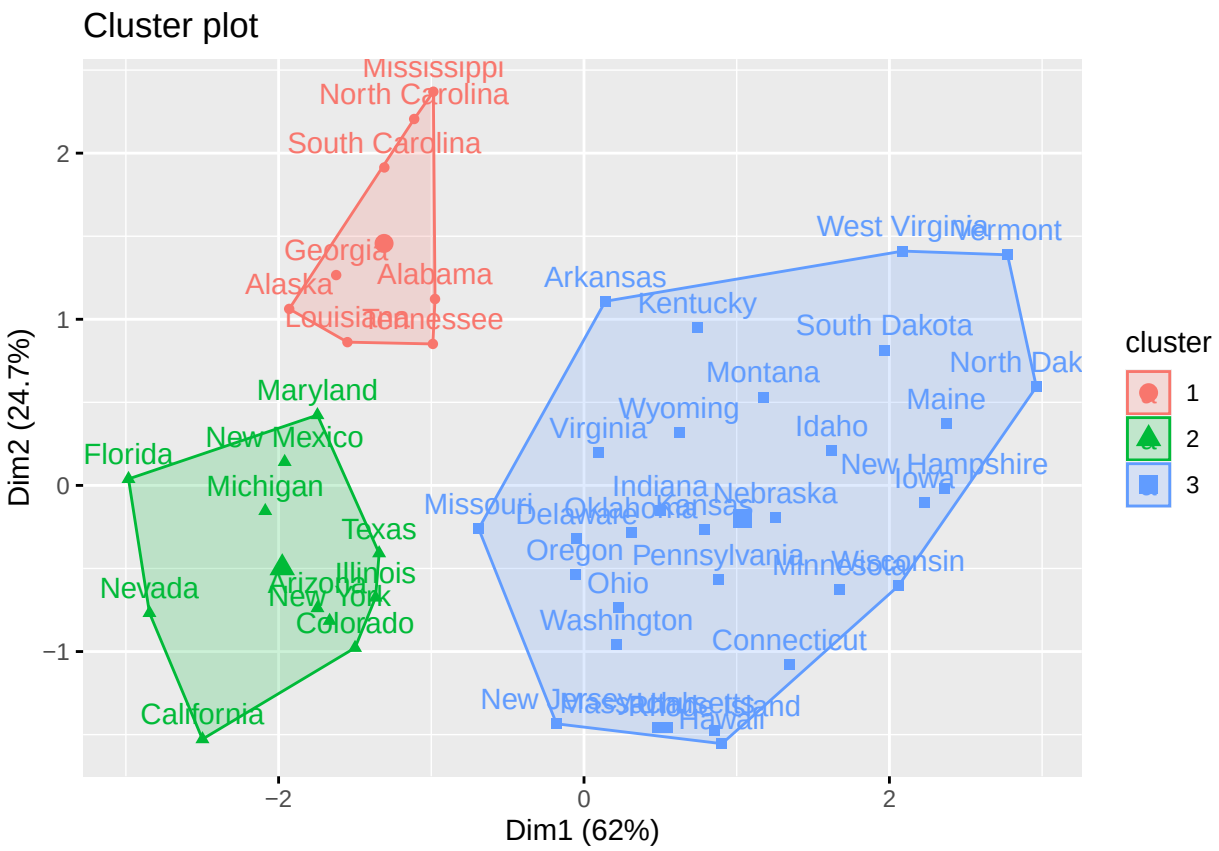
Cluster 3 (Azul) Ohio, Utah, Connecticut, Pensilvania, Nebraska, Kentucky, Montana, Idaho, Indiana, Kansas, Hawái, Minnesota, Wisconsin, Iowa, Nuevo Hampshire, Virginia Occidental, Maine, Dakota del Sur, Dakota del Norte y Vermont.

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

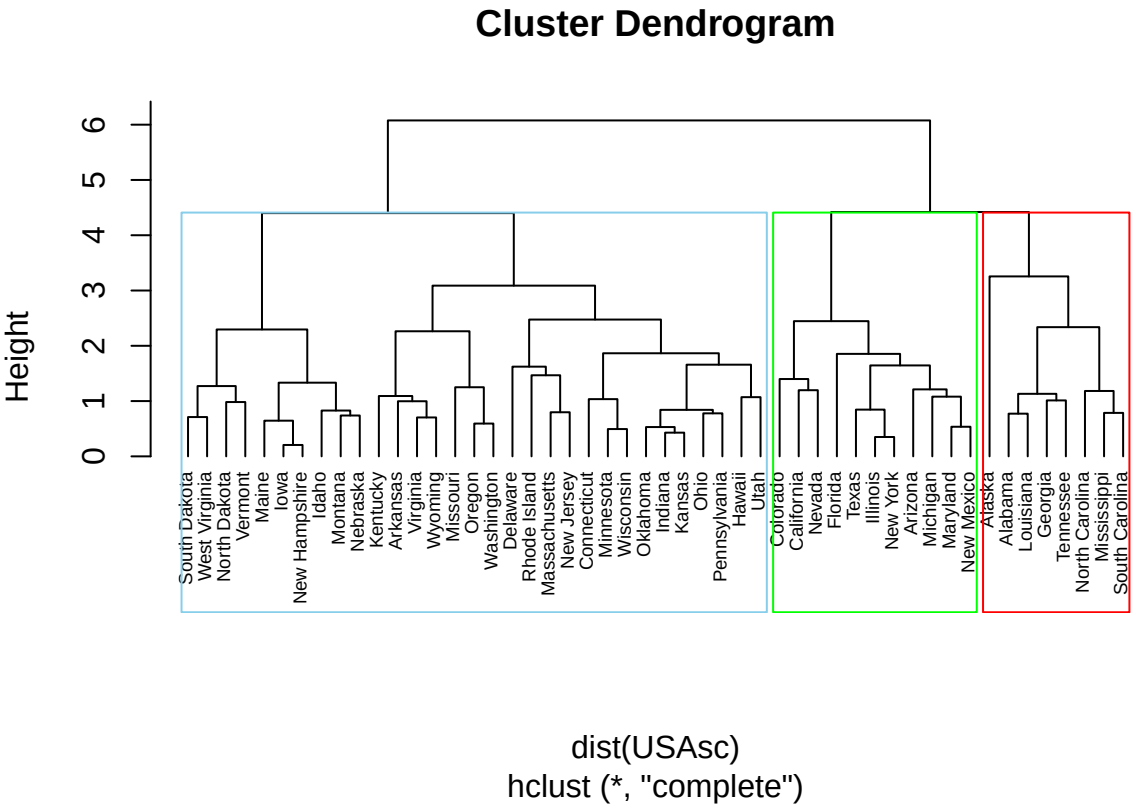
```
USAsc=scale(USArrests)
hc.scale = hclust(dist(USAsc), method ="complete")
plot(hc.scale, main ="Hierarchical Clustering with Scaled Features", xlab="", sub ="",
cex =0.7)
```




```
clust <- cutree(hc.scale, k = 3)
fviz_cluster(list(data = USAsc, cluster = clust))
```




```
plot(hc.scale, hang = -1, cex = 0.6)
rect.hclust(hc.scale, k = 3, border = c("skyblue","green","red"))
```



Observando el anterior dendograma, en el cual se escalo los datos para que las variables tuvieran magnitudes parecidas (alturas menores), se escogen tres cluster como resultado, cada cluster le pertenecen los siguientes estados.

Cluster 1 (Azul) Dakota del Sur, Virginia Occidental, Dakota del Norte, Vermont, Maine, Iowa, Nueva Hampshire, Idaho, Montana, Nebraska, Kentucky, Arkansas, Virginia, Wyoming, Delaware, Rhode Island, Massachusetts, Nueva Jersey, Connecticut, Minnesota, Wisconsin, Oklahoma, Indiana, Kansas, Ohio, Pensilvania, Hawái y Utah.

Cluster 2 (Verde) Colorado, California, Nevada, Florida, Texas, Illionois, Nueva York, Arizona, Michigan, Maryland Y Nuevo México.

Cluster 3 (Rojo) Alaska, Alabama, Luisiana, Georgia, Tennessee, Norte de California, Misisipi y Sur de California.

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.

Al escalar las variables se tiene una media de cero y una desviación estándar de uno, esto se hace para que las magnitudes de las varianzas no puedan afectar en gran medida los resultados obtenidos por el **clustering**, por ejemplo en este caso la varibale **Asalto** tiene mayor media y varianza, que las demas variables.

Al comparar los dos dendogramas obtenidos en **b)** y en **c)**, se puede resaltar que el dendograma al escalar las variables tiene una altura menor, ya que las magnitudes de las variables son más parecidas, además, los 3 cluster cambiaron la pertenencia de los estados, dando a entender que la variable **Assault** afectaba las disimilitudes del dendograma de **b)**.

Con lo anterior se opina que si es necesario escalar las variables antes de que se calculen las disimilitudes, ya que sino se escala; las variables que tengan mayor media y varianza determinará en gran medida la distancia obtenida al comparar observaciones, dirigiendo así la agrupación final; además que el escalar permite que todas las variables tengan el mismo peso para realizar el **clustering**.