UNIVERSIDAD NACIONAL DE COLOMBIA

Trabajo 2

Luis David Hernández Pérez

Daniel Felipe Villa Rengifo

Juan Gabriel Carvajal Negrete

Said Alejandro Duran

Introducción a análitica

Cesar Augusto Gomez Velez

Universidad Nacional de Colombia

Sede Medellín

Facultad de Ciencias

Medellín, Noviembre del 2023

Tabajo 2: Modulo 2

1. (**30Pts**) En este ejercicio se utilizarán arboles de regresión para predecir los valores de la variable sales en la base de datos Carseats de la libreria ISLR2,tratando dicha variable como continua:

   a) Divida el conjunto de observaciones en un conjunto de entrenamiento y un conjunto de prueba. De forma aleatoria. En que proporciones dividió los datos?.

   b) Ajuste un árbol de regresión en el conjunto de entrenamiento. Grafique arbol e interprete los resultados. Que valor del MSE **de prueba** obtiene?.

   c) Utilice validación cruzada para determinar el grado óptimo de complijidad del árbol. Consigue la poda del árbol mejorar el EMS **de prueba**?

   d) Utilice el método **bagging** para analizar estos datos. Que valor del MSE obtiene?

      - Use la funcion importance() para determinar cúal de las variables es la más importante.

   e) Ahora utilice un **bosque aleatorio** (Random-Forest) para analizar estos datos.

      - Que valor del MSE de **prueba** obtiene?.

      - Use la función importance para determinar cuales variables son las más importantes.

      - Describa el efecto de m el número de variables consideradas en cada subdivisióon, en la tasa de error obtenida.

Solución

Punto 1:

a) División de la base de datos

Para dividir la base de datos tomaremos el 70% de las observaciones para entrenamiento y el otro 30% para prueba.

```
set.seed(0.21)
datos <- ISLR2::Carseats
lenp <- floor(0.7*nrow(datos))
filas_train <- sample(1:nrow(datos),lenp)
datos_train <- datos[filas_train,]
datos_test <- datos[-filas_train,]
```

Se presenta los primeros y últimos registros con algunas de las variables de la base de datos de entrenamiento y de prueba, esto para intentar minimizar espacio en el documento.

Datos de entrenamiento

ID	Sales	CompPrice	Income	Education	Urban	US
398	7.41	162	26	18	Yes	Yes
324	10.36	107	105	12	Yes	Yes
167	6.71	119	67	11	Yes	Yes
129	4.96	133	100	13	Yes	Yes
...	...	...	...	...		
309	9.24	126	80	10	Yes	Yes
109	3.47	107	79	16	Yes	No
361	8.77	118	86	15	No	Yes
74	12.61	118	90	11	No	Yes

Datos de prueba

ID	Sales	CompPrice	Income	Education	Urban	US
3	10.06	113	35	12	Yes	Yes
4	7.4	117	100	14	Yes	Yes
5	4.15	141	64	13	Yes	No
6	10.81	124	113	16	No	Yes
...	...	...	...	...		
386	5.87	131	73	17	Yes	Yes
392	6.1	153	63	16	Yes	No
393	4.53	129	42	13	Yes	Yes
399	5.94	100	79	12	Yes	Yes

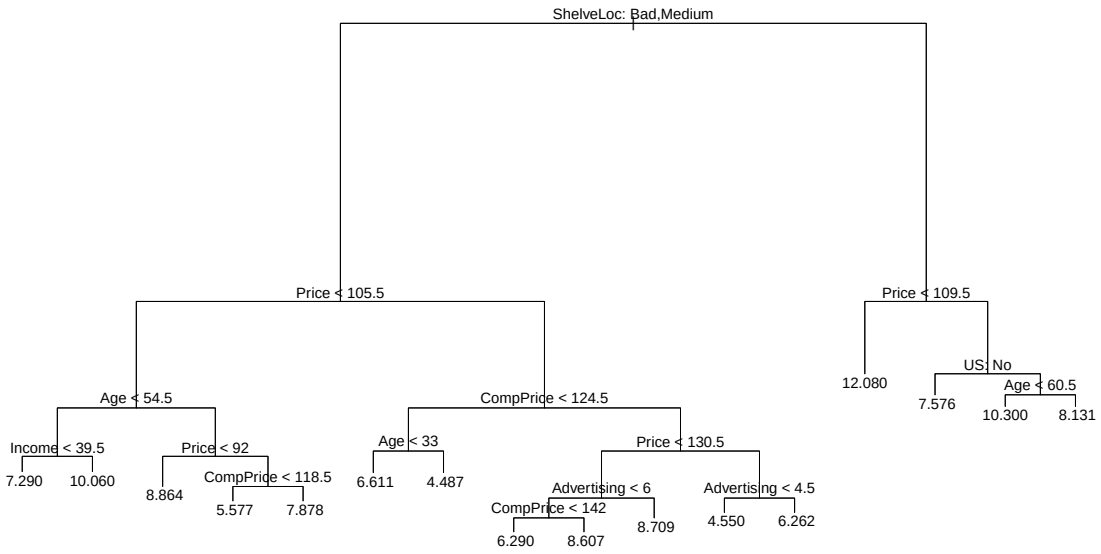
b) Árbol de regresión ajustados a los datos de entrenamiento

Con ayuda de la función tree() que es la encargada de realizar arboles de decisión y de regresión en R, se ajusta el modelo a el conjunto de datos de entrenamiento obtenidos de la base de datos **Carseats** de la librería ISLR2

```
tree.Carseats <- tree::tree(Sales~.,data = datos_train)
```

Gráfica del árbol de regresión

```
plot(tree.Carseats)
text(tree.Carseats ,pretty =0)
```



La variable categórica Shelveloc(Calidad de la ubicación de los estándares bad,good,medium) es la mas determinante para determinar las ventas unitarias de sillas de carros para niños y si los estándares son calificados como “bad” tienden a tener menos ventas que en las otras dos categorías. El árbol muestra que para la categoría Medium de la variable Shelveloc las variables CompPrice,Income,Advertising,Population,Education,Urban,Us parecen tener jugar poco papel en las ventes individuales.

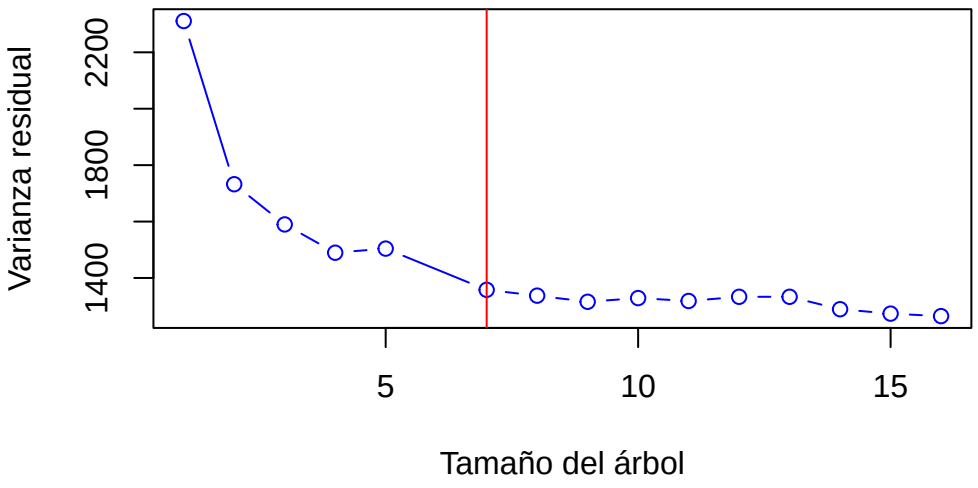
```
yhat <- predict(tree.Carseats ,newdata = datos_test)
MSE <- mean((datos_test$Sales-yhat)^2)
```

El MSE de prueba obtenido del modelo de árbol de regresión ajustado con los datos de entrenamiento y evaluado con los datos de test es **5.1147485**

c) Validación cruzada para determinar el grado óptimo

La validación cruzada para determinar el grado optimo de complejidad del árbol se realiza con la funcion **cv.tree()**

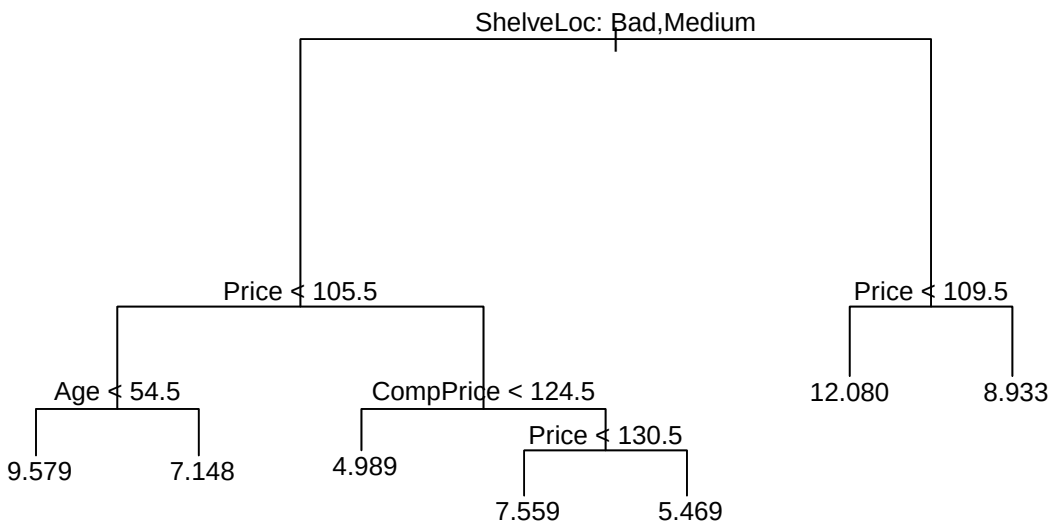
```
cv.Carseats <- cv.tree(tree.Carseats)
plot(cv.Carseats$size ,cv.Carseats$dev,type='b',col="blue",
     xlab = "Tamaño del árbol",ylab = "Varianza residual")
abline(v=7, col = "red")
```



Según la gráfica de los procesos de validación cruzada para determinar el tamaño óptimo del árbol, el número de áreas que minimiza la varianza residual es de 16 áreas. Sin embargo, dado que buscamos reducir el tamaño del árbol a tal punto que la varianza residual no cambie considerablemente a medida que agregamos más áreas, elegimos como tamaño óptimo del árbol 7 áreas.

Ahora se poda el árbol con ayuda de la función prune.tree() e indicamos que el mejor valor para el numero de áreas del árbol es 7.

```
prune.Carseats <- prune.tree(tree.Carseats,best = 7)
plot(prune.Carseats)
text(prune.Carseats ,pretty =0)
```



Revisemos si el proceso de la determinación del tamaño optimo del árbol redujo el MSE de prueba del modelo ajustado.

```
yhat.P <- predict(prune.Carseats ,newdata = datos_test)
MSE.P <- mean((datos_test$Sales-yhat.P)^2)
```

Efectivamente podar el árbol de regresión inicial reduce el MSE de prueba de nuestro modelo a un 4.695608

d) Método bagging

Con ayuda de la función randomForest() de la librería randomForest ajustemos un modelo bagged a los datos de entrenamiento y evaluemos que tan bien se desempeña el model.

El parámetro mtry nos sirve para indicar cuantas variables se quieren utilizar en cada split o segmentación en la construcción del árbol.

```
set.seed(0.021)
bag.Carseats <- randomForest(Sales~.,data=datos_train,mtry=10, importance =TRUE)
bag.Carseats
```

```
Call:
 randomForest(formula = Sales ~ ., data = datos_train, mtry = 10,      importance = TRUE)
```

```
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 10

          Mean of squared residuals: 2.653509
                    % Var explained: 67.35
```

```r
# Valor MSE
yhat.bag <- predict(bag.Carseats,newdata = datos_test)
MSE.bag <- mean((datos_test$Sales-yhat.bag)^2)
```

Evaluando nuestro modelo con los datos de prueba se optiene un MSE de 2.4428133

- Las variables mas importantes que identifica nuestro modelo son las siguientes.

Variable	%IncMSE	IncNodePurity
CompPrice	34.11	237.85
Income	10.23	120.45
Advertising	22.04	162.33
Population	-0.99	62.44
Price	70.23	665.47
ShelveLoc	65.64	638.67
Age	22.03	244.49
Education	2.74	61.01
Urban	-2.18	11.21
US	3.99	14.47

e) Bosque aleatorio (Random-Forest)

Generar un bosque aleatorio procede exactamente de la misma manera, excepto que se utiliza un valor menor del argumento mtry. Por defecto, randomForest() utiliza p/3 variables al construir un bosque aleatorio de árboles de regresión, y $\sqrt{p}$ variables al construir un bosque aleatorio de árboles de regresión bosotros utilizaremos **mtry**=$\sqrt{10}$.

```r
set.seed(0.0021)
rf.Carseats <- randomForest(Sales~.,data=datos_train,mtry = sqrt(10), importance =TRUE)
rf.Carseats
```

```
Call:
 randomForest(formula = Sales ~ ., data = datos_train, mtry = sqrt(10),      importance = TRUE
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 3.030197
                    % Var explained: 62.71
```

```
yhat.rf <-  predict(rf.Carseats ,newdata = datos_test)
MSE.rf <- mean(( yhat.rf - datos_test$Sales)^2)
```

- El valor del MSE de **prueba** obtenido es de 2.9226249
- La importancia de las variables dentro del modelo

Variable	%IncMSE	IncNodePurity
CompPrice	19.79	218.51
Income	4.35	162.95
Advertising	16.54	186.10
Population	-1.15	128.04
Price	41.10	525.19
ShelveLoc	44.68	500.83
Age	18.14	274.38
Education	1.85	103.00
Urban	-1.54	20.72
US	6.86	41.69

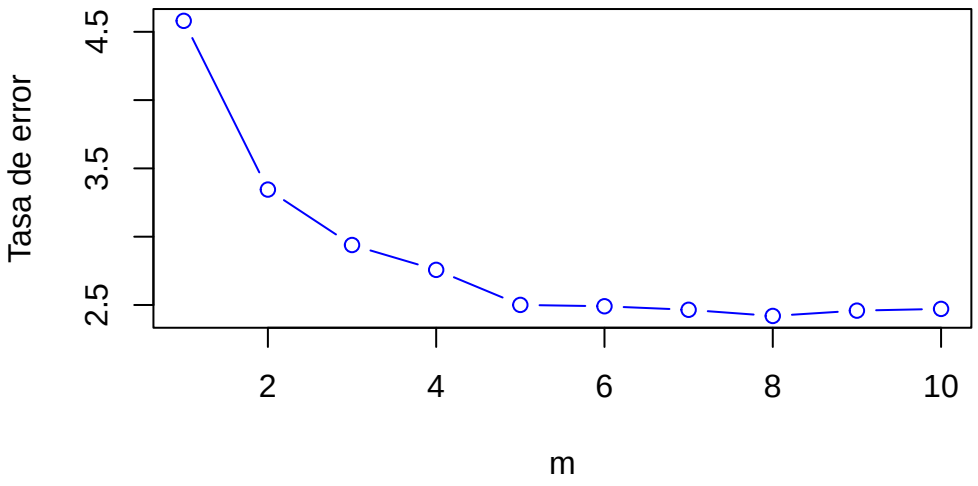
La segunda columna está basada en la disminución media de la precisión en las predicciones en las muestras bagging cuando una variable dada se excluye del modelo. La tercera columna es una medida de la disminución total en la impureza del nodo que resulta de divisiones sobre esa variable, promediada sobre todos los árboles.

- Efecto de m en la tasa de error obtenida

```
Tasa_error <- c()
for (i in 1:10){
  mod <- randomForest(Sales~.,data = datos_train,mtry=i,
                      importance=TRUE)
  pre <- predict(mod,newdata=datos_test)
  Tasa_error[i] <- mean((datos_test$Sales-pre)^2)
}
```

Los MSE de prueba para los diferentes valores de m de 1 a 10 en el modelo de randomforest son los siguientes 4.5803129, 3.3449841, 2.9390684, 2.7572168, 2.5006194, 2.4907467, 2.4649431, 2.4197444, 2.4587539, 2.4712735 respectivamente

```
plot(1:10,Tasa_error,type = "b",xlab = "m",ylab = "Tasa de error",
     col="blue")
```



El gráfico anterior muestra cómo influye el valor de m en el cálculo del MSE de **prueba** del modelo de árbol aleatorio. Cuando m toma valores de 5 en adelante, la tasa de error parece no variar demasiado su valor. Gracias a la gráfica, podemos identificar que un buen valor para m podría ser 5.

```
set.seed(0.0021)
rf.Carseats <- randomForest(Sales~.,data=datos_train,mtry = 5,
                            importance =TRUE)
yhat.rf <-  predict(rf.Carseats ,newdata = datos_test)
MSE.rf <- mean(( yhat.rf - datos_test$Sales)^2)
```

Con un valor de MSE de prueba de 2.5726373

2. **(30Pts)** Este ejercicio utiliza el conjunto de datos OJ el cual es parte de lalibrería ISLR2.

a) Cree un conjunto de entrenamiento con una muestra aletoria de 800 ob- servaciones y un conjunto de prueba que conste del resto de observaciones.

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

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

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

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

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

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

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

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

Solución

Punto 2:

a) División de la base de datos

De la base de datos tomaremos aleatoriamente un conjunto de entrenamiento de 800 observaciones y un conjunto de prueba de 270 observaciones.

```
set.seed(0.00021)
datos1 <- ISLR2::OJ
datos <- ISLR2::Carseats
filas_train1 <- sample(1:nrow(datos1),800)
datos_train1 <- datos1[filas_train1,]
datos_test1 <- datos1[-filas_train1,]
```

Se presenta los primeros y últimos registros con algunas de las variables de la base de datos de entrenamiento y de prueba, esto para intentar minimizar espacio en el documento.

Datos de entrenamiento

ID	Purchase	WeekofPurchase	StoreID	PctDiscMM	PctDiscCH	ListPriceDiff
1017	CH	236	7	0.2	0	0.24
679	CH	277	1	0	0.12	0.14
129	CH	267	2	0.18	0	0.32
930	MM	236	3	0	0	0.3
...		...	...	...	...	...
627	CH	274	7	0.25	0.25	0.27
525	MM	262	1	0.2	0	0.23
833	MM	237	3	0	0	0.3
332	MM	266	1	0.05	0	0.13

Datos de prueba

ID	Purchase	WeekofPurchase	StoreID	PctDiscMM	PctDiscCH	ListPriceDiff
2	CH	239	1	0.15	0	0.24
4	MM	227	1	0	0	0
10	CH	238	7	0.2	0	0.24
18	MM	268	2	0	0	0.32
...		...	...	...	...	...
1045	CH	241	1	0.15	0	0.13
1049	CH	256	1	0	0	0.42
1051	CH	230	7	0	0	0.3
1060	CH	235	1	0	0	0.3

b) Clasificador de soporte vectorial

Para ajustar un clasificador de soporte vectorial se usa la función svm() de la librería e1071, se usa un kernel lineal y se usa 0.1 como el valor del parámetro **cost** que es el costo de una violación de la margen. El modelo intentará predecir los valores de la variable **Pursachase** basada en las demás variables de la base de datos.

```
svm.oj <- svm(Purchase~., data=datos_train1 , kernel ="linear", cost = 0.1,scale = FALSE )
svm.oj %>% summary()
```

```
Call:
svm(formula = Purchase ~ ., data = datos_train1, kernel = "linear",
    cost = 0.1, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  432

 ( 216 216 )


Number of Classes:  2

Levels:
 CH MM
```

Del resumen del modelo indica que este es de tipo clasificación (C-classification), también muestra el tipo de kernel (lineal),el valor del parámetro cost 0.1. El modelo utilizo 432 vectores de soporte 216 en la clase CH y 216 en la clase MM.

c) Tasas de error de entrenamiento y de prueba.

Tasas de error de entrenamiento

```
y.pred.t <- predict(svm.oj,newdata = datos_train1)
matriz.train <- table(predict=y.pred.t,truth=datos_train1$Purchase)
FP <- matriz.train[2, 1]  # Falsos positivos
FN <- matriz.train[1, 2]  # Falsos negativos
TP <- matriz.train[1, 1]  # Verdaderos positivos
TN <- matriz.train[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train)
```

La tasa de error de entrenamiento obtenida es de 0.17125 lo que significa que el modelo a la hora de clasificar se equivoca aproximadamente en el 17% de las predicciones en el conjunto de datos evaluado.En otras palabras,83% de las predicciones fueron correctas.

Tasas de error de prueba

```
y.pred.p <- predict(svm.oj,newdata = datos_test1)
matriz.prueba <- table(predict=y.pred.p,truth=datos_test1$Purchase)
FP <- matriz.prueba[2, 1]  # Falsos positivos
FN <- matriz.prueba[1, 2]  # Falsos negativos
TP <- matriz.prueba[1, 1]  # Verdaderos positivos
TN <- matriz.prueba[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.l <- (FP + FN) / sum(matriz.prueba)
```

La tasa de error de prueba obtenida es de 0.17125 lo que significa que el modelo a la hora de clasificar se equivoca aproximadamente en el 20% de las predicciones en el conjunto de datos evaluado.En otras,80% de las predicciones fueron correctas.

d) Valor óptimo de cost.

La librería e1071 incluye una función incorporada, tune(), para realizar validación cruzada. Por defecto, tune() realiza una validación cruzada de diez folds en un conjunto de modelos de interés.Para utilizar esta función, se pasa información relevante sobre el conjunto de modelos que se están considerando. El siguiente comando indica que se quiere comparar SVM con un lineal kernel, utilizando un rango de valores del parámetro cost.

```
set.seed(0.000021)
tune.out <- tune(svm,Purchase~.,data = datos_train1,kernel="linear",
                 ranges = list(cost=c(0.01,0.5,1,5,10)))
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.5

- best performance: 0.17125

- Detailed performance results:
   cost   error dispersion
1  0.01 0.17250 0.03574602
2  0.50 0.17125 0.03488573
3  1.00 0.17125 0.03537988
```

```
4  5.00 0.17375 0.03557562
5 10.00 0.17500 0.03818813
```

El valor que genera el menor error de validación cruzada es **cost**=0.5 .

e) Tasas de error de entrenamineto y prueba esta vez con el valor optimo de cost.

Para acceder al mejor modelo la función tune() guarda dicho modelo, para acceder a el se hace de la siguiente manera.

```r
best.svm.oj <- tune.out$best.model
summary(best.svm.oj)
```

```
Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = datos_train1,
    ranges = list(cost = c(0.01, 0.5, 1, 5, 10)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.5

Number of Support Vectors:  329

 ( 165 164 )


Number of Classes:  2

Levels:
 CH MM
```

Ahora calculemos las tasas de error de entrenamiento y de prueba de este modelo.

Tasa de error de entrenamiento

```r
y.pred.t1 <- predict(best.svm.oj,newdata = datos_train1)
matriz.train.b <- table(predict=y.pred.t1,truth=datos_train1$Purchase)
FP <- matriz.train.b[2, 1]  # Falsos positivos
FN <- matriz.train.b[1, 2]  # Falsos negativos
TP <- matriz.train.b[1, 1]  # Verdaderos positivos
TN <- matriz.train.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train.b)
```

La tasa de error es de 0.16625.El modelo se equivoca aproximadamente el 16% de las predicciones en el conjunto de datos evaluado, en comparación con la tasa de error de entrenamiento con el modelo ajustado con el parámetro **cost**= 0.1, este modelo mejora en un 1% las predicciones correctas.

Tasa de error de prueba

```
y.pred.p1 <- predict(best.svm.oj,newdata = datos_test1)
matriz.test.b <- table(predict=y.pred.p1,truth=datos_test1$Purchase)
FP <- matriz.test.b[2, 1]  # Falsos positivos
FN <- matriz.test.b[1, 2]  # Falsos negativos
TP <- matriz.test.b[1, 1]  # Verdaderos positivos
TN <- matriz.test.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.L <- (FP + FN) / sum(matriz.test.b)
```

Para la tasa de error de prueba se obtiene un valor de 0.1666667 muy parecida a ña tasa de error de entrenamiento y en comparación con el modelo ajustado con el valor de 0.01 para el parámetro **cost**, este modelo mejora considerablemente las predicciones con un 84% de predicciones correctas.

e) Maquina de soporte vectorial.

Ahora se ajusta una maquina de porte vectorial con un núcleo radial y utilizando el valor default para γ.

```
svm.oj.M <- svm(Purchase~.,data = datos_train1,kernel="radial",cost=0.1)
summary(svm.oj.M)
```

```
Call:
svm(formula = Purchase ~ ., data = datos_train1, kernel = "radial",
    cost = 0.1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.1

Number of Support Vectors:  538

 ( 271 267 )


Number of Classes:  2

Levels:
 CH MM
```

Del resumen del modelo indica que este es de tipo clasificación (C-classification), también muestra el tipo de kernel (radial),el valor del parámetro cost 0.1. El modelo utilizo 538 vectores de soporte, 271 en la clase CH y 267 en la clase MM.

Tasa de error de entrenamiento.

```
y.pred.t.m <- predict(svm.oj.M,newdata = datos_train1)
matriz.train.m <- table(predict=y.pred.t.m,truth=datos_train1$Purchase)
FP <- matriz.train.m[2, 1]  # Falsos positivos
FN <- matriz.train.m[1, 2]  # Falsos negativos
TP <- matriz.train.m[1, 1]  # Verdaderos positivos
TN <- matriz.train.m[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train.m)
```

El modelo tiene una tasa de error de entrenamiento de 0.16375.El modelo se equivoca aproximadamente en un 16% de las predicciones con el conjunto de entrenamiento.

Tasa de error de prueba.

```
y.pred.p.m <- predict(svm.oj.M,newdata = datos_test1)
matriz.test.m <- table(predict=y.pred.p.m,truth=datos_test1$Purchase)
FP <- matriz.test.m[2, 1]  # Falsos positivos
FN <- matriz.test.m[1, 2]  # Falsos negativos
TP <- matriz.test.m[1, 1]  # Verdaderos positivos
TN <- matriz.test.m[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.r <- (FP + FN) / sum(matriz.test.m)
```

La tasa de error de prueba es de 0.16375.El modelo a la hora de clasificar se equivoca en un 22% de las predicciones sobre el conjunto de datos de prueba.

valor optimo del parámetro cost.

```
tune.out.M <-  tune(svm , Purchase~., data=datos_train1, kernel = "radial",
ranges =list(cost=c(0.01,0.5,1,5,10) ))
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.5

- best performance: 0.17125
```

```
- Detailed performance results:
   cost   error dispersion
1  0.01 0.17250 0.03574602
2  0.50 0.17125 0.03488573
3  1.00 0.17125 0.03537988
4  5.00 0.17375 0.03557562
5 10.00 0.17500 0.03818813
```

Según el resumen de la función tune el valor de cost que reduce el error de la validación es de 0.5. Calculemos las tasas de error del modelo.

Tasa de error de entrenamiento.

```r
svm.oj.M.b <- tune.out.M$best.model
y.pred.t.m.b <- predict(svm.oj.M.b,newdata = datos_train1)
matriz.train.m.b <- table(predict=y.pred.t.m.b,truth=datos_train1$Purchase)
FP <- matriz.train.m.b[2, 1]  # Falsos positivos
FN <- matriz.train.m.b[1, 2]  # Falsos negativos
TP <- matriz.train.m.b[1, 1]  # Verdaderos positivos
TN <- matriz.train.m.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train.m.b)
```

Se obtuvo un valor de 0.14625 para la tasa de error de entrenamiento, lo que quiere decir que el modelo se equivoca aproximadamente en un 14% de a la hora de clasificar las observaciones de los datos de entrenamiento.

Tasa de error de prueba.

```r
y.pred.p.m.b <- predict(svm.oj.M.b,newdata = datos_test1)
matriz.test.m.b <- table(predict=y.pred.p.m.b,truth=datos_test1$Purchase)
FP <- matriz.test.m.b[2, 1]  # Falsos positivos
FN <- matriz.test.m.b[1, 2]  # Falsos negativos
TP <- matriz.test.m.b[1, 1]  # Verdaderos positivos
TN <- matriz.test.m.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.R <- (FP + FN) / sum(matriz.test.m.b)
```

La tasa de error de prueba es de 0.1888889. Aproximadamente el 18% de las observaciones han sido erróneamente calificadas por el modelo.

e) Maquina de soporte vectorial con nucleo polinomial.

A continuación se ajusta una maquina de soporte vectorial con núcleo polinomial usando degree=2.

```r
svm.oj.M1 <- svm(Purchase~.,data =datos_train1,kernel="polynomial",cost=0.1,degree=2)
summary(svm.oj.M1)
```

```
Call:
svm(formula = Purchase ~ ., data = datos_train1, kernel = "polynomial",
    cost = 0.1, degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.1
     degree:  2
     coef.0:  0

Number of Support Vectors:  582

 ( 296 286 )


Number of Classes:  2

Levels:
 CH MM
```

Del resumen del modelo indica que este es de tipo clasificación (C-classification), también muestra el tipo de kernel (polynomial),el valor del parámetro cost 0.1,degree= 2. El modelo utilizo 582 vectores de soporte 296 en la clase CH y 286 en la clase MM.

Tasa de error de entrenamiento.

```r
y.pred.t.m1 <- predict(svm.oj.M1,newdata = datos_train1)
matriz.train.m1 <- table(predict=y.pred.t.m1,truth=datos_train1$Purchase)
FP <- matriz.train.m1[2, 1]  # Falsos positivos
FN <- matriz.train.m1[1, 2]  # Falsos negativos
TP <- matriz.train.m1[1, 1]  # Verdaderos positivos
TN <- matriz.train.m1[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train.m1)
```

La tasa de error de entrenamiento es de 0.295, es decir el modelo se equivoca aproximadamente un 29% de las predicciones.

Tasa de error de prueba.

```r
y.pred.p.m1 <- predict(svm.oj.M1,newdata = datos_test1)
matriz.test.m1 <- table(predict=y.pred.p.m1,truth=datos_test1$Purchase)
```

```r
FP <- matriz.test.m1[2, 1]  # Falsos positivos
FN <- matriz.test.m1[1, 2]  # Falsos negativos
TP <- matriz.test.m1[1, 1]  # Verdaderos positivos
TN <- matriz.test.m1[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.p <- (FP + FN) / sum(matriz.test.m1)
```

La tasa de error de prueba fue de 0.295, aproximadamente el 30% de las observaciones han sido erróneamente calificadas.

optimo del parametro cost

```r
tune.out.M1 <-  tune(svm , Purchase~., data=datos_train1, kernel = "polynomial",
ranges =list(cost=c(0.01,0.5,1,5,10) ),degree=2)
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.5

- best performance: 0.17125

- Detailed performance results:
   cost   error dispersion
1  0.01 0.17250 0.03574602
2  0.50 0.17125 0.03488573
3  1.00 0.17125 0.03537988
4  5.00 0.17375 0.03557562
5 10.00 0.17500 0.03818813
```

El mejor modelo es aquel que tiene 0.5 como valor del parametro cost.

Tasa de error de entrenamiento

```r
svm.oj.M1.b <- tune.out.M1$best.model
y.pred.t.m1.b <- predict(svm.oj.M1.b,newdata = datos_train1)
matriz.train.m1.b <- table(predict=y.pred.t.m1.b,truth=datos_train1$Purchase)
FP <- matriz.train.m1.b[2, 1]  # Falsos positivos
FN <- matriz.train.m1.b[1, 2]  # Falsos negativos
TP <- matriz.train.m1.b[1, 1]  # Verdaderos positivos
TN <- matriz.train.m1.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error <- (FP + FN) / sum(matriz.train.m1.b)
```

La tasa de error de entrenamiento es de 0.15. aproximadamente el 13% de las observaciones de entrenamiento han sido erróneamente clasificada.

Tasa de error de prueba

```
y.pred.p.m1.b <- predict(svm.oj.M1.b,newdata = datos_test1)
matriz.prueba.m1.b <- table(predict=y.pred.p.m1.b,truth=datos_test1$Purchase)
FP <- matriz.prueba.m1.b[2, 1]  # Falsos positivos
FN <- matriz.prueba.m1.b[1, 2]  # Falsos negativos
TP <- matriz.prueba.m1.b[1, 1]  # Verdaderos positivos
TN <- matriz.prueba.m1.b[2, 2]  # Verdaderos negativos
# Calcular la tasa de error de entrenamiento
tasa_error.P <- (FP + FN) / sum(matriz.prueba.m1.b)
```

La tasa de prueba es de 0.2074074 lo que indica que el modelo predice correctamente aproximadamente el 79% de las observaciones del conjunto de prueba.

NOTA: En los tres modelos implementados la malla de valores para el parámetro cost de función **tune** coincide, esto se debe a que no se utilizaron muchos valores para ser evaluados en la validación cruzada.

e) Mejor modelo

Para determinar cual es el kernel que hace que la maquina de soporte vectorial funcione mejor para los datos, vamos a utilizar la tasa de error de prueba luego de haber determinado el valor optimo del parámetro cost. Vale la pena resaltar que los modelos mejoran la tasa de predicciones siempre que se les determina el mejor valor para cost

```
tabla <- matrix(c(tasa_error.l,tasa_error.L,tasa_error.r,tasa_error.R,tasa_error.p,
                  tasa_error.P),ncol = 3,nrow = 2)
colnames(tabla) <- c("Liner","Radial","polynomial")
rownames(tabla) <- c("cost=0.1","cost=0.5")
tabla %>% xtable()
```

	Tasa de error		
Cost	**Linear**	**Radial**	**Polynomial**
0.1	**0.2037037**	**0.2222222**	**0.3037037**
0.5	**0.1703704**	**0.1888889**	**0.2148148**

El mejor modelo que se ajusta a los datos es el clasificador de soporte vectorial (kernel=linear,cost=0.5).

3. **(40Pts)** [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(); Otra opción es runif().Asegúrese de agregar un **cambio en la media** en las observaciones de cada clase a fin de obtener tres clases distintas.

   b) Realice PCA en las 60 observaciones y grafique las observaciones en téminos de las 2 primeras variables principales Z1 y textcolorredZ2. **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 simulacióon 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.

   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.

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

   e) Ahora realice agrupamiento de K-medias con K = 4 y describa su resultados.

   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 agrupacion de K-medias en la matriz de tamaño 60×2, cuya primera columna es la coordenada z_{i1} en la primera componente principal Z1 y la segunda columna es la coordenada z_{i2} en la segunda componente principal textcolorredZ2. **Comente los resultados**.

   g) Con la funcion 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

Punto 3:

a)

Utilizaremos la función `rnorm()` para generar los datos.

```r
set.seed(123)

# Número de observaciones por clase
num_obs <- 20

# Número de variables
num_variables <- 50

# Generar datos para la primera clase con media 0
class1 <- matrix(rnorm(num_obs * num_variables, mean = 0, sd = 1), nrow = num_obs)

# Generar datos para la segunda clase con media 2
class2 <- matrix(rnorm(num_obs * num_variables, mean = 2, sd = 1), nrow = num_obs)

# Generar datos para la tercera clase con media -2
class3 <- matrix(rnorm(num_obs * num_variables, mean = -2, sd = 1), nrow = num_obs)

# Combinar los datos de las tres clases
simulated_data <- rbind(class1, class2, class3)

# Crear un vector de clases (1, 2, 3)
# para identificar a qué clase pertenece cada observación
clases <- rep(1:3, each = num_obs)

# Agregar el vector de clases como una columna a los datos simulados
simulated_data_with_classes <- cbind(clases, simulated_data)

# Convertir la matriz en un data frame para facilitar el manejo
simulated_data_df <- as.data.frame(simulated_data_with_classes)
```

Literal (b)

Primero examinamos brevemente los datos.

```r
apply(simulated_data_df[,-51], 2, FUN = mean) %>% round(2)
```

```
clases      V2      V3      V4      V5      V6      V7      V8      V9     V10     V11
  2.00    0.13    0.12    0.05   -0.13   -0.02   -0.14   -0.05   -0.15   -0.09    0.00
   V12     V13     V14     V15     V16     V17     V18     V19     V20     V21     V22
  0.02   -0.09   -0.08    0.14    0.04   -0.01    0.00   -0.18    0.16   -0.02   -0.20
   V23     V24     V25     V26     V27     V28     V29     V30     V31     V32     V33
  0.02    0.11    0.00    0.34    0.03    0.16    0.01    0.07   -0.10   -0.07    0.13
   V34     V35     V36     V37     V38     V39     V40     V41     V42     V43     V44
 -0.07   -0.01   -0.27   -0.04    0.03    0.12    0.08    0.01    0.01   -0.05    0.15
   V45     V46     V47     V48     V49     V50
  0.22    0.01    0.08   -0.17    0.22    0.00
```

De la salida anterior notamos que las variables poseen medias muy diferentes.

También podemos examinar las varianzas.

```
apply(simulated_data_df[, -51], 2, FUN = var) %>% round(2)
```

```
clases      V2      V3      V4      V5      V6      V7      V8      V9     V10     V11
  0.68    4.31    3.63    3.40    4.04    4.36    3.51    4.01    3.76    3.81    3.73
   V12     V13     V14     V15     V16     V17     V18     V19     V20     V21     V22
  2.98    3.39    4.63    4.09    3.56    3.29    3.71    3.78    3.32    3.76    4.45
   V23     V24     V25     V26     V27     V28     V29     V30     V31     V32     V33
  3.93    3.88    3.06    3.93    3.42    3.63    4.03    2.83    3.55    4.70    3.88
   V34     V35     V36     V37     V38     V39     V40     V41     V42     V43     V44
  3.28    4.35    4.30    4.54    2.91    4.39    3.78    3.23    3.51    3.85    3.64
   V45     V46     V47     V48     V49     V50
  4.56    3.49    4.48    3.87    3.43    3.62
```

De la salida anterior también se puede observar que las variables tienen varianzas muy diferentes.

Ahora realizaremos el análisis de componentes principales utilizando la función `prcomp()`, la cual es una de varias funciones en R que realizan PCA.

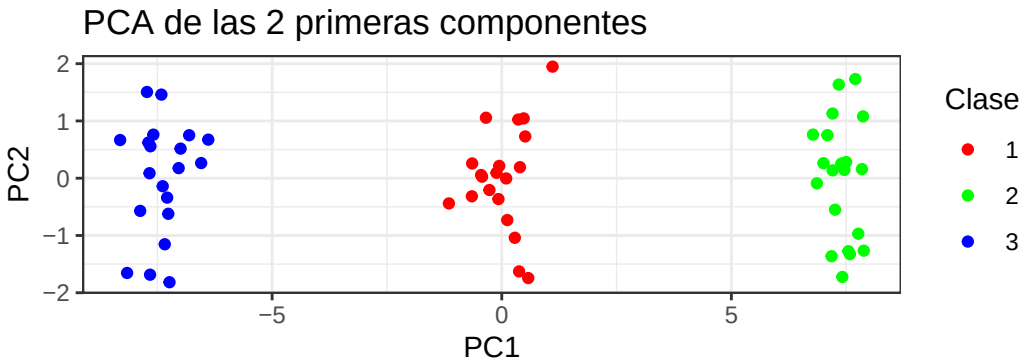


Figura 1: Gráfico de las dos primeras componentes

Observamos en el gráfico anterior Figura 1 tres clases distintas, cada una representada por un color diferente: rojo para la clase 1, verde para la clase 2 y azul para la clase 3. La distribución de los puntos sugiere que el PCA ha logrado una separación relativa entre las clases, especialmente entre la clase 3 y las otras dos clases, en las direcciones de las dos primeras componentes principales.

Literal (c)

```
# Realizar agrupación de K-means, con k=3
set.seed(123)
kmeans_result <- kmeans(simulated_data_df[, -1], centers = 3, nstart = 20)

comparison_table <- table("Clase Verdadera" = simulated_data_df$clases,
                          "Cluster Asignado" = kmeans_result$cluster)
comparison_table
```

```
               Cluster Asignado
Clase Verdadera  1  2  3
              1  0 20  0
              2 20  0  0
              3  0  0 20
```

La tabla indica que cada una de las clases verdaderas ha sido asignada a un cluster distinto, sin mezclas entre ellas. Esto significa que los 20 elementos de la clase verdadera 1 han sido asignados al cluster 2, los 20 elementos de la clase verdadera 2 al cluster 1, y los 20 elementos de la clase verdadera 3 al cluster 3.

El hecho de que no haya mezcla entre las clases en los clusters asignados por K-medias sugiere que el algoritmo ha realizado una perfecta clasificación de los datos en base a sus características, asignando cada observación a un cluster correspondiente a su clase verdadera. La interpretación debe tener en cuenta que los números de los clusters asignados por K-medias son arbitrarios y no tienen por qué coincidir con las etiquetas de las clases verdaderas, por lo que una coincidencia directa no es necesaria para que la clasificación sea correcta.

Literal (d)

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

```
# Realizar agrupación de K-means, con k=2
set.seed(123)
kmeans_result <- kmeans(simulated_data_df[, -1], centers = 2, nstart = 20)

comparison_table <- table("Clase Verdadera" = simulated_data_df$clases,
                          "Cluster Asignado" = kmeans_result$cluster)
comparison_table
```

```
                Cluster Asignado
Clase Verdadera  1  2
              1  0 20
              2  0 20
              3 20  0
```

Este resultado indica que el algoritmo de K-medias ha agrupado juntas las Clases Verdaderas 1 y 2 en el mismo cluster (Cluster 2), mientras que la Clase Verdadera 3 ha sido separada en un cluster diferente (Cluster 1). Dado que K-medias con $K = 2$ solo permite dos grupos, el algoritmo ha combinado las dos clases más similares entre sí en términos de la distancia de sus características.

Literal (e)

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

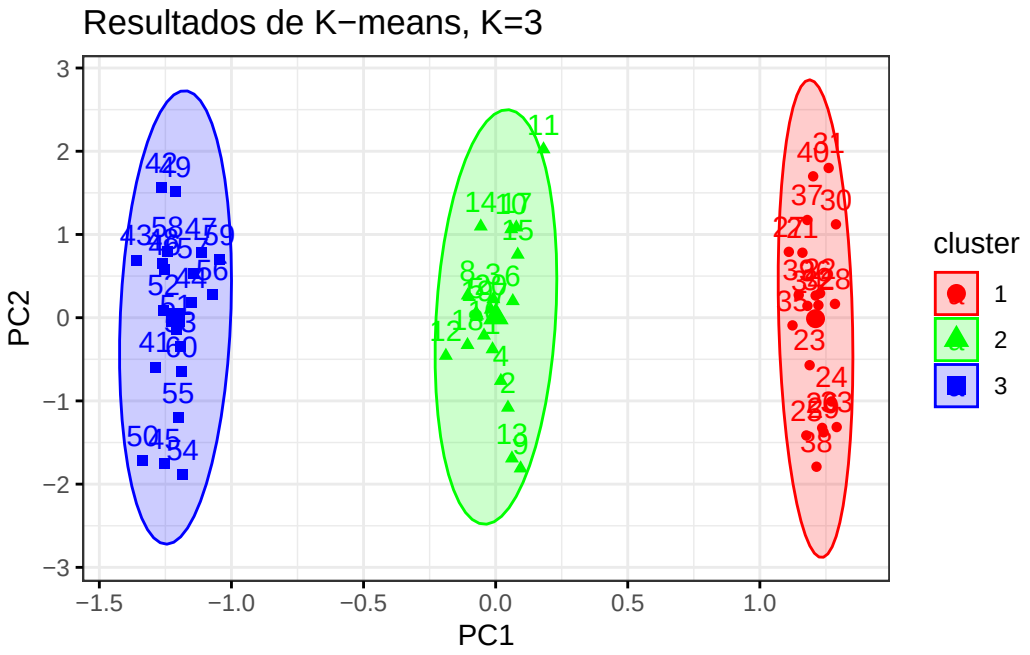
```r
# Realizar agrupación de K-means, con k=4
set.seed(123)
kmeans_result <- kmeans(simulated_data_df[, -1], centers = 4, nstart = 20)

comparison_table <- table("Clase Verdadera" = simulated_data_df$clases,
                          "Cluster Asignado" = kmeans_result$cluster)
comparison_table
```

```
                Cluster Asignado
Clase Verdadera  1  2  3  4
              1  0  0 20  0
              2  8  0  0 12
              3  0 20  0  0
```

Este resultado sugiere que mientras las Clases Verdaderas 1 y 3 han sido identificadas y segregadas claramente en clusters únicos (Clusters 3 y 2 respectivamente), la Clase Verdadera 2 ha sido repartida entre dos clusters. Esto puede indicar que, desde la perspectiva del algoritmo y basado en la medida de distancia utilizada (usualmente la distancia euclidiana), las observaciones de la Clase Verdadera 2 no son tan cohesivas como las de las Clases Verdaderas 1 y 3, y quizás presentan una mayor variabilidad interna o tienen características que las solapan con las variabilidades de las otras clases.

El hecho de que haya una división en la Clase Verdadera 2 podría reflejar una estructura subyacente no capturada por las etiquetas de clase originales o podría ser una señal de que el valor de K escogido no es el más adecuado para este conjunto de datos. En la práctica, elegir el valor correcto de K a menudo implica un compromiso entre sobre simplificar la estructura de los datos (un valor de K muy bajo) y sobreajustar el ruido (un valor de K muy alto). Herramientas como el método del codo o la silueta pueden ayudar a determinar el valor más apropiado de K.

Literal (f)



El gráfico ilustra cómo se distribuyen las observaciones en el espacio de las dos primeras componentes principales (PC1 y PC2), que son combinaciones lineales de las variables originales, seleccionadas de tal manera que maximizan la varianza de los datos. Al aplicar K-means a las componentes principales en lugar de las variables originales, se está agrupando los datos basándose en las direcciones de mayor varianza, lo cual puede llevar a una mejor separación de los clusters si las componentes principales capturan bien la estructura de los datos.

Es de resaltar que la separación clara entre los clusters sugiere que el agrupamiento ha sido efectivo en este espacio de componentes principales. Esto es especialmente útil si las variables originales tienen una correlación alta o si el conjunto de datos es de alta dimensión, ya que el PCA reduce la redundancia antes de aplicar K-medias, lo que puede mejorar la calidad de los clusters encontrados. Además, trabajar con componentes principales en lugar de con las variables originales puede hacer que la interpretación de los resultados sea más sencilla y más resistente a ciertos tipos de ruido y variabilidad irrelevante en los datos.

Literal (g)

```r
# Escalar los datos
scaled_data <- scale(simulated_data_df[, -1])

# Realizar agrupación de K-medias en los datos escalados
set.seed(123)
kmeans_scaled_result <- kmeans(scaled_data, centers = 3)

comparison_table_scaled <- table("Clase Verdadera" = simulated_data_df$clases,
                                 "Cluster Asignado" = kmeans_scaled_result$cluster)

comparison_table_scaled
```

```
               Cluster Asignado
Clase Verdadera  1  2  3
              1  0 20  0
              2 20  0  0
              3  0  0 20
```

Comparando estos resultados con los obtenidos en (b), podemos observar que en ambos casos el algoritmo de K-medias logró un agrupamiento perfecto. Sin embargo, la diferencia clave entre los dos métodos es el enfoque de la transformación de los datos:

- el caso de la PCA, se realizó una reducción de dimensionalidad, donde se transformaron las variables originales en componentes principales que capturan la mayor variación posible en los datos.
- En el caso de la estandarización con `scale()`, se ajustó la escala de las variables originales para que todas contribuyeran equitativamente al análisis de K-medias, eliminando el sesgo que podrían introducir las variables con mayor varianza.

La consistencia de los resultados entre los dos métodos sugiere que las clases verdaderas tienen diferencias significativas que se pueden detectar tanto en las direcciones de máxima varianza (PCA) como en el espacio de las variables originales después de la estandarización. Esto también indica que las clases son inherentemente bien separables y que los métodos de preprocesamiento aplicados (PCA y estandarización) han sido efectivos para revelar esta separabilidad en el espacio de características transformado.