Tarea 2 - 1EST17

Tabla de Contenidos

Grupo

- **Integrante**: Lucio Enrique Cornejo Ramírez
- **Código**: 20192058

Problema 1

```
library(rpart)
```

```
Warning: package 'rpart' was built under R version 4.1.3
```

```
library(party)
```

```
Warning: package 'party' was built under R version 4.1.3
```

```
Loading required package: grid
```

```
Loading required package: mvtnorm
```

```
Loading required package: modeltools
```

```
Loading required package: stats4
```

```
Loading required package: strucchange
```

```
Warning: package 'strucchange' was built under R version 4.1.3
```

```
Loading required package: zoo
```

```
Warning: package 'zoo' was built under R version 4.1.3
```

```
Attaching package: 'zoo'
```

```
The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

```
Loading required package: sandwich
```

```
Warning: package 'sandwich' was built under R version 4.1.3
```

```r
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.1.3
```

Caso particular

```r
# Tamaño de muestra
n <- 100

datos <- data.frame(
  X1 = runif(n, 0, 1),
  X2 = runif(n, 0, 1),
  X3 = runif(n, 0, 1),
  X4 = sample(1:4, n, replace = TRUE),
  Y = rbinom(n, size = 1, prob = 0.5)
)

# Variables categóricas
datos$X4 <- as.factor(datos$X4)
datos$Y <- as.factor(datos$Y)

head(datos, 20)
```

```
          X1         X2        X3 X4 Y
1  0.69353069 0.88984119 0.9766679  2 1
2  0.78581326 0.82842698 0.1443011  3 1
3  0.76950164 0.45800623 0.9397259  2 0
4  0.61424740 0.70474228 0.6807788  2 1
5  0.05595438 0.41970234 0.7292801  2 0
6  0.58563907 0.01366936 0.4311478  3 1
7  0.43252858 0.60448523 0.5584109  3 0
8  0.19111180 0.46082231 0.9018560  3 0
9  0.34897113 0.30590440 0.3996011  4 0
10 0.31003843 0.17443229 0.9876088  2 1
11 0.59260299 0.88521774 0.5372602  3 1
12 0.21777472 0.88858409 0.2963797  2 0
13 0.19650612 0.35646302 0.1168249  2 1
14 0.92323810 0.39364182 0.7124939  4 1
15 0.04396489 0.28522933 0.7147088  1 1
16 0.12186171 0.72035836 0.6358998  2 0
```

```
17 0.77367649 0.38472382 0.3313009  3 1
18 0.26223229 0.33861888 0.1223218  3 0
19 0.93068913 0.74138040 0.9930549  4 1
20 0.62673141 0.17845827 0.5600250  2 1
```

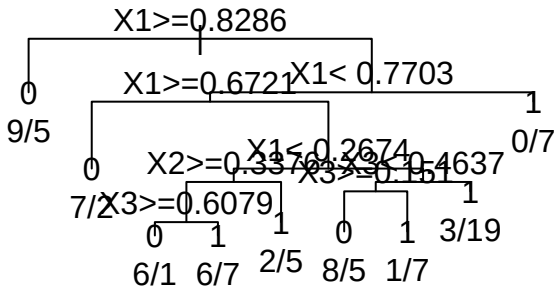
```r
r_arbol <- rpart(
  Y ~ ., data = datos, method = 'class', cp = 0
)
r_arbol
```

```
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 100 42 1 (0.4200000 0.5800000)
    2) X1>=0.8285608 14  5 0 (0.6428571 0.3571429) *
    3) X1< 0.8285608 86 33 1 (0.3837209 0.6162791)
      6) X1< 0.7703289 79 33 1 (0.4177215 0.5822785)
       12) X1>=0.6720678 9  2 0 (0.7777778 0.2222222) *
       13) X1< 0.6720678 70 26 1 (0.3714286 0.6285714)
         26) X1< 0.2674443 27 13 0 (0.5185185 0.4814815)
           52) X2>=0.337618 20  8 0 (0.6000000 0.4000000)
            104) X3>=0.6079051 7  1 0 (0.8571429 0.1428571) *
            105) X3< 0.6079051 13  6 1 (0.4615385 0.5384615) *
           53) X2< 0.337618 7  2 1 (0.2857143 0.7142857) *
         27) X1>=0.2674443 43 12 1 (0.2790698 0.7209302)
           54) X3< 0.4636989 21  9 1 (0.4285714 0.5714286)
            108) X3>=0.1510146 13  5 0 (0.6153846 0.3846154) *
            109) X3< 0.1510146 8  1 1 (0.1250000 0.8750000) *
           55) X3>=0.4636989 22  3 1 (0.1363636 0.8636364) *
      7) X1>=0.7703289 7  0 1 (0.0000000 1.0000000) *
```

```r
plot(r_arbol, margin = 0.25)
text(r_arbol, use.n = TRUE)
```




```
c_arbol <- ctree(
  Y ~ ., data = datos,
  controls = ctree_control(mincriterion = 0)
)
c_arbol
```

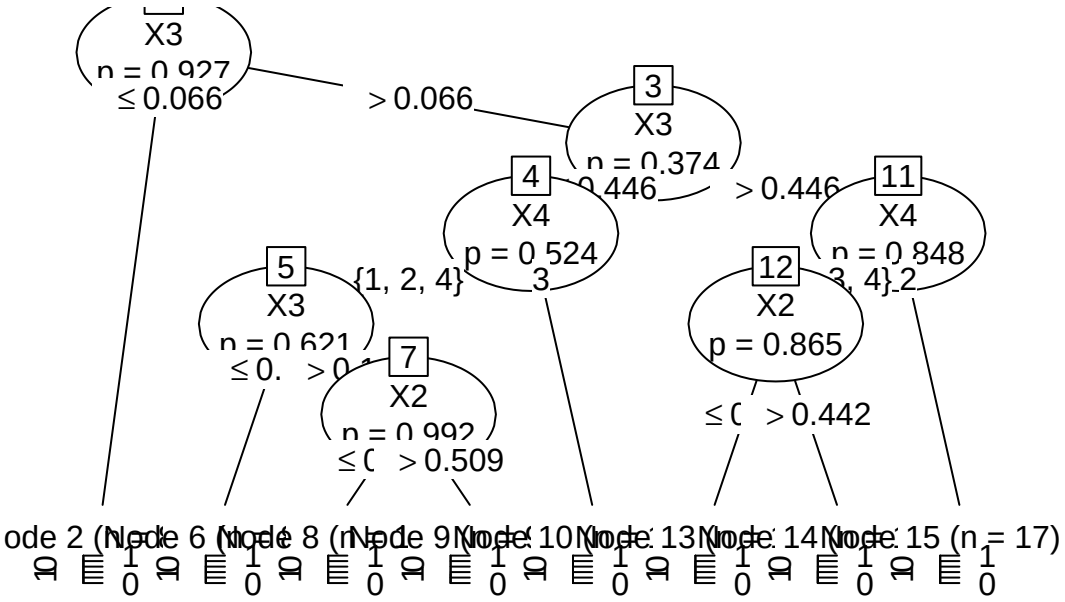
```
     Conditional inference tree with 8 terminal nodes

Response:  Y
Inputs:  X1, X2, X3, X4
Number of observations:  100

1) X3 <= 0.06613136; criterion = 0.073, statistic = 0.726
  2)*  weights = 8
1) X3 > 0.06613136
  3) X3 <= 0.4464696; criterion = 0.626, statistic = 2.548
    4) X4 == {1, 2, 4}; criterion = 0.476, statistic = 5.036
      5) X3 <= 0.1452606; criterion = 0.379, statistic = 1.536
        6)*  weights = 8
      5) X3 > 0.1452606
        7) X2 <= 0.5091768; criterion = 0.008, statistic = 0.689
          8)*  weights = 14
        7) X2 > 0.5091768
          9)*  weights = 9
    4) X4 == {3}
      10)*  weights = 13
  3) X3 > 0.4464696
    11) X4 == {1, 3, 4}; criterion = 0.152, statistic = 3.109
      12) X2 <= 0.442387; criterion = 0.135, statistic = 0.728
        13)*  weights = 13
      12) X2 > 0.442387
        14)*  weights = 18
```

```
  11) X4 == {2}
    15)*  weights = 17
```

```
plot(c_arbol)
```



Simulación

```
var.rpart <- function(fit) {
  tabla <- table(fit$frame$var)[-1]
  sol <- as.numeric(tabla > 0)
  names(sol) <- names(tabla)
  return(sol)
}

var.ctree <- function(fit){
  require(gdata)
  a <- capture.output(print(fit))
  a <- a[-c(1:7)]
  a <- trim(a)

  v <- character()
  for(h in 1:length(a)){
    b <- a[h]
```

```
    b <- gsub(") ","q", b)
    b <- gsub(")", "q", b)
    b <- gsub(" < ", "q", b)
    b <- gsub(" > ", "q", b)
    b <- gsub(" <= ", "q", b)
    b <- gsub(" >= ", "q", b)
    b <- gsub(" weights = ", "q", b)
    v[h] <- unlist(strsplit(b, "q"))[2]
  }

  v <- factor(
    v, levels = names(fit@data@get("input"))
  )
  v <- v[is.na(v)==F]
  tabla <- table(v) > 0
  sol <- as.numeric(table(v) > 0)
  names(sol) <- names(tabla)
  return(sol)
}

# Funciones auxiliares
random_df <- function(n = 100) {
  # n: Tamaño de muestra
  datos <- data.frame(
    X1 = runif(n, 0, 1),
    X2 = runif(n, 0, 1),
    X3 = runif(n, 0, 1),
    X4 = sample(1:4, n, replace = TRUE),
    Y = rbinom(n, size = 1, prob = 0.5)
  )

  # Variables categóricas
  datos$X4 <- as.factor(datos$X4)
  datos$Y <- as.factor(datos$Y)

  return(datos)
}

r_tree <- function(df) {
  rpart(Y ~ ., data = df, method = 'class', cp = 0)
}
```

```r
c_tree <- function(df) {
  ctree(Y ~ ., data = df, controls = ctree_control(mincriterion = 0))
}
```

```r
num_simul <- 10**4
```

```r
r_vars_frequency <- rep(0, 4)
names(r_vars_frequency) <- paste0("X", 1:4)

c_vars_frequency <- rep(0, 4)
names(c_vars_frequency) <- paste0("X", 1:4)

set.seed(6174)
for (simul_n in 1:num_simul) {
  df <- random_df()

  tree <- r_tree(df)
  # plot(tree)
  # text(tree, use.n = TRUE)
  vars <- var.rpart(tree)
  for (var_name in names(vars)) {
    r_vars_frequency[var_name] <- 1 + r_vars_frequency[var_name]
  }

  tree <- c_tree(df)
  # plot(tree)
  vars <- var.ctree(tree)
  for (var_name in names(vars)) {
    c_vars_frequency[var_name] <- 1 + c_vars_frequency[var_name]
  }
}
```

```
Loading required package: gdata

Attaching package: 'gdata'

The following object is masked from 'package:stats4':

    nobs
```

```
The following object is masked from 'package:stats':

    nobs

The following object is masked from 'package:utils':

    object.size

The following object is masked from 'package:base':

    startsWith
```

```r
# Frecuencia con que cada covariable fue seleccionada
r_vars_frequency
```

```
  X1   X2   X3   X4
8447 8323 8238 5758
```

```r
c_vars_frequency
```

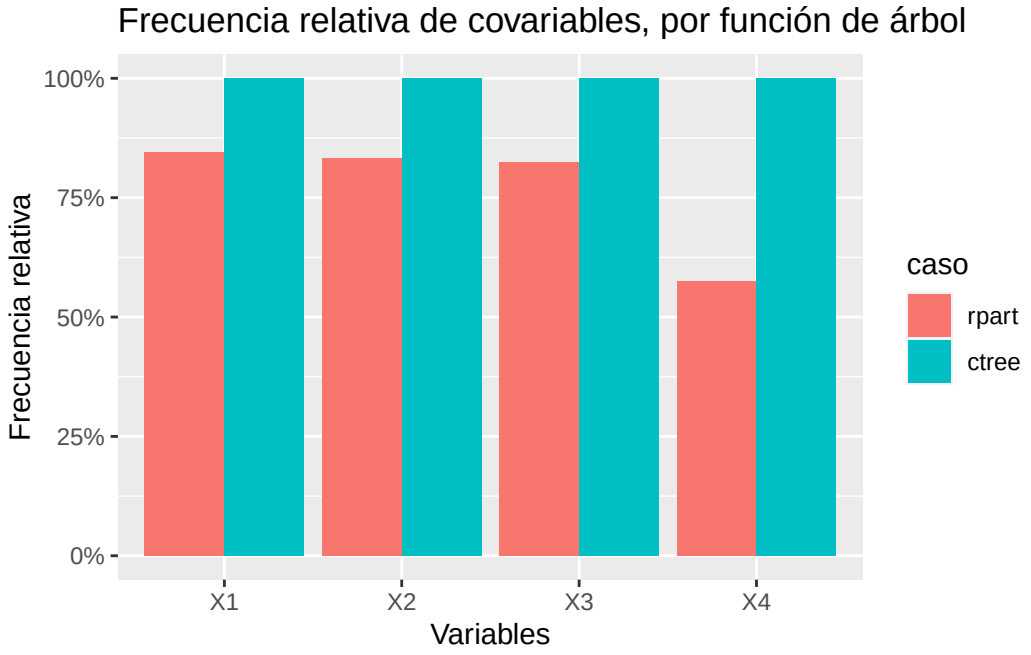
```
   X1    X2    X3    X4
10000 10000 10000 10000
```

```r
# Gráfico de las frecuencias por covariables
vars_frequencies <- data.frame(
  variables = paste0("X", 1:4),
  rpart_freq = r_vars_frequency,
  ctree_freq = c_vars_frequency
)

vars_frequencies <- rbind(
  data.frame(
    variables = paste0("X", 1:4),
    frecuencia = unname(r_vars_frequency),
    caso = as.factor("rpart")
  ),
  data.frame(
    variables = paste0("X", 1:4),
    frecuencia = unname(c_vars_frequency),
```

```
    caso = as.factor("ctree")
  )
)

ggplot(vars_frequencies) +
  aes(x = variables, y = frecuencia / num_simul, fill = caso) +
  geom_bar(stat = 'identity', position = 'dodge') +
  scale_y_continuous(labels = scales::percent) +
  labs(
    x = "Variables", y = "Frecuencia relativa",
    title = "Frecuencia relativa de covariables, por función de árbol"
  )
```



Respuesta

En base a la simulación presentada, estimamos que, para los **dos** tipos empleados de construcción de un árbol (funciones `r_tree` y `c_tree`), la **probabilidad de que una covariable sea seleccionada** es de:

Caso	X1	X2	X3	X4
`rpart`	0.8447	0.8323	0.8238	0.5758
`ctree`	1	1	1	1

En base a que las probabilidades en la segunda fila de la tabla previa son todas del mismo valor (1, en particular), concluimos a favor de la afirmación de *Hothorn et al.*. Esto debido a que la ausencia del sesgo a la hora de seleccionar una covariable es evidente, comparado a como se observa en la primera fila de la tabla previa, donde se observa un **menor sesgo** por escoger la covariable $X4$, variable categórica con una menor cantidad de posibles puntos de corte, comparado a variables numéricas.

Problema 2

```
library(e1071)
```

```
Warning: package 'e1071' was built under R version 4.1.3
```

```
library(ISLR2)
```

```
Warning: package 'ISLR2' was built under R version 4.1.3
```

```
data(OJ)

str(OJ)
```

```
'data.frame':   1070 obs. of  18 variables:
 $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
 $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
 $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
 $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
 $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
 $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
 $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
 $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
 $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
 $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
 $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
 $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
 $ PctDiscMM     : num  0 0.151 0 0 0 ...
 $ PctDiscCH     : num  0 0 0.0914 0 0 ...
 $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
 $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

Parte a)

```r
# Separamos datos en conjuntos de entrenamiento y de prueba
set.seed(4268)
train_id <- sample(1:nrow(OJ), 800)
train <- OJ[train_id,]
test <- OJ[-train_id,]
```

Parte b)

```r
svmfit <- svm(
  Purchase ~ ., data = train, kernel = "linear", cost = 0.01
)
summary(svmfit)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  431

 ( 217 214 )


Number of Classes:  2

Levels:
 CH MM
```

Note que la cantidad de vectores de soporte, 431, representa aproximadamente el 50% de las observaciones en los datos con los que fue entrenado el modelo.

Este porcentaje relativamente elevado es un **posible indicador** de que un kernel lineal **no es apropiado** para la separación de los datos vía un hiperplano.

Parte c)

```r
# Función auxiliar para matriz de confusión y tasa de error
confusion_matrix_y_error <- function(df, svm.model) {
  confusion_matrix <- table(
    true = df$Purchase, pred = predict(svm.model, df[,-1])
  )

  error_rate <-
    (confusion_matrix[1, 2] + confusion_matrix[2, 1]) / nrow(df)

  return(list(tabla = confusion_matrix, error = error_rate))
}
```

```r
tmp <- confusion_matrix_y_error(train, svmfit)
train_confusion_matrix_b <- tmp[["tabla"]]
train_confusion_matrix_b
```

```
    pred
true  CH  MM
  CH 431  56
  MM  78 235
```

```r
train_error_b <- tmp[["error"]]
train_error_b
```

```
[1] 0.1675
```

```r
tmp <- confusion_matrix_y_error(test, svmfit)
test_confusion_matrix_b <- tmp[["tabla"]]
test_confusion_matrix_b
```

```
    pred
true  CH  MM
  CH 143  23
  MM  25  79
```

```r
test_error_b <- tmp[["error"]]
test_error_b
```

```
[1] 0.1777778
```

Las tasas de error de entrenamiento y prueba son, respectivamente, 16.75 % y 17.7777778 % .

Parte d)

```r
set.seed(4268)
tune.costo = tune(
  svm, Purchase ~., data = train, kernel = "linear",
  ranges = list(cost = seq(0.01, 10, length.out = 20))
)
summary(tune.costo)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 8.948421

- best performance: 0.16375

- Detailed performance results:
         cost   error dispersion
1   0.0100000 0.16875 0.03830162
2   0.5357895 0.16625 0.04251225
3   1.0615789 0.17000 0.03827895
4   1.5873684 0.16875 0.03875224
5   2.1131579 0.16750 0.04005205
6   2.6389474 0.16625 0.03955042
7   3.1647368 0.16500 0.04199868
8   3.6905263 0.16625 0.03955042
9   4.2163158 0.16625 0.03955042
10  4.7421053 0.16625 0.03955042
11  5.2678947 0.16875 0.03784563
```

```
12  5.7936842 0.16875 0.03784563
13  6.3194737 0.16875 0.03784563
14  6.8452632 0.16875 0.03784563
15  7.3710526 0.16625 0.03998698
16  7.8968421 0.16625 0.03998698
17  8.4226316 0.16625 0.03998698
18  8.9484211 0.16375 0.03928617
19  9.4742105 0.16375 0.03928617
20 10.0000000 0.16375 0.03928617
```

Respecto a los veinte casos evaluados para el valor de `costo`, el valor encontrado de **costo óptimo** es de `8.948421`.

Parte e)

```r
# Seleccionamos el mejor (según costo) modelo encontrado
svmfit_d <- tune.costo$best.model
svmfit_d
```

```
Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = seq(0.01,
    10, length.out = 20)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  8.948421

Number of Support Vectors:  323
```

```r
tmp <- confusion_matrix_y_error(train, svmfit_d)
train_confusion_matrix_d <- tmp[["tabla"]]
train_confusion_matrix_d
```

```
    pred
true  CH  MM
  CH 434  53
  MM  73 240
```

```r
train_error_d <- tmp[["error"]]
train_error_d
```

```
[1] 0.1575
```

```r
tmp <- confusion_matrix_y_error(test, svmfit_d)
test_confusion_matrix_d <- tmp[["tabla"]]
test_confusion_matrix_d
```

```
    pred
true  CH  MM
  CH 143  23
  MM  23  81
```

```r
test_error_d <- tmp[["error"]]
test_error_d
```

```
[1] 0.1703704
```

Para el *mejor* modelo encontrado en la parte `d)`, las tasas de error de entrenamiento y prueba son, respectivamente, 15.75 % y 17.037037 % .

Parte `f)`

Parte `f.b)`

Evitaremos especificar el valor de `gamma`, en la función `svm`, con el fin de usar su **valor predeterminado**.

```r
svmfit_f <- svm(
  Purchase ~ ., data = train, kernel = "radial", cost = 0.01
)
summary(svmfit_f)
```

```
Call:
svm(formula = Purchase ~ ., data = train, kernel = "radial", cost = 0.01)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01

Number of Support Vectors:  629

 ( 316 313 )


Number of Classes:  2

Levels:
 CH MM
```

Respecto a este nuevo modelo, con kernel radial, encontramos que el número de vectores de soporte (629) representa aproximadamente el 80% de las observaciones de los datos de entrenamiento.

En ese sentido, respecto al valor de costo especificado, podríamos inferir que aquel modelo radial no es tan apropiado (incluso menos apropiado que el modelo lineal en la parte b)) para clasificación de observaciones.

Parte f.c)

```r
tmp <- confusion_matrix_y_error(train, svmfit_f)
train_confusion_matrix_f.b <- tmp[["tabla"]]
train_confusion_matrix_f.b
```

```
    pred
true  CH  MM
  CH 487   0
  MM 313   0
```

```r
train_error_f.b <- tmp[["error"]]
train_error_f.b
```

```
[1] 0.39125
```

```r
tmp <- confusion_matrix_y_error(test, svmfit_f)
test_confusion_matrix_f.b <- tmp[["tabla"]]
test_confusion_matrix_f.b
```

```
    pred
true  CH  MM
  CH 166   0
  MM 104   0
```

```r
test_error_f.b <- tmp[["error"]]
test_error_f.b
```

```
[1] 0.3851852
```

Las tasas de error de entrenamiento y prueba son, respectivamente, 39.125 % y 38.5185185 % .

Parte `f.d)`

```r
set.seed(4268)
tune.costo_f = tune(
  svm, Purchase ~., data = train, kernel = "radial",
  ranges = list(cost = seq(0.01, 10, length.out = 20))
)
summary(tune.costo_f)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 3.164737

- best performance: 0.1625

- Detailed performance results:
         cost   error dispersion
```

```
1   0.0100000 0.39125 0.06320436
2   0.5357895 0.16625 0.03682259
3   1.0615789 0.16500 0.03944053
4   1.5873684 0.16250 0.03952847
5   2.1131579 0.16375 0.03884174
6   2.6389474 0.16500 0.04241004
7   3.1647368 0.16250 0.03818813
8   3.6905263 0.16500 0.03944053
9   4.2163158 0.16750 0.04048319
10  4.7421053 0.16875 0.03963812
11  5.2678947 0.17000 0.04005205
12  5.7936842 0.17125 0.03998698
13  6.3194737 0.17000 0.04005205
14  6.8452632 0.17125 0.04084609
15  7.3710526 0.17250 0.04199868
16  7.8968421 0.17375 0.04016027
17  8.4226316 0.17500 0.04208127
18  8.9484211 0.17625 0.04059026
19  9.4742105 0.17750 0.03987829
20 10.0000000 0.17625 0.04059026
```

Respecto a los veinte casos evaluados para el valor de `costo`, el valor encontrado de **costo óptimo** es de `3.164737`.

Parte `f.e)`

```
# Seleccionamos el mejor (según costo) modelo encontrado
svmfit_f.d <- tune.costo_f$best.model
svmfit_f.d
```

```
Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = seq(0.01,
    10, length.out = 20)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  3.164737

Number of Support Vectors:  333
```

```
tmp <- confusion_matrix_y_error(train, svmfit_f.d)
train_confusion_matrix_f.d <- tmp[["tabla"]]
train_confusion_matrix_f.d
```

```
    pred
true  CH  MM
  CH 450  37
  MM  73 240
```

```
train_error_f.d <- tmp[["error"]]
train_error_f.d
```

```
[1] 0.1375
```

```
tmp <- confusion_matrix_y_error(test, svmfit_f.d)
test_confusion_matrix_f.d <- tmp[["tabla"]]
test_confusion_matrix_f.d
```

```
    pred
true  CH  MM
  CH 146  20
  MM  28  76
```

```
test_error_f.d <- tmp[["error"]]
test_error_f.d
```

```
[1] 0.1777778
```

Para el *mejor* modelo encontrado en la parte `d)`, las tasas de error de entrenamiento y prueba son, respectivamente, 13.75 % y 17.7777778 % .

Parte `g)`

Parte `g.b)`

```r
svmfit_g <- svm(
  Purchase ~ ., data = train, cost = 0.01,
  kernel = "polynomial", degree = 2
)
summary(svmfit_g)
```

```
Call:
svm(formula = Purchase ~ ., data = train, cost = 0.01, kernel = "polynomial",
    degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2
     coef.0:  0

Number of Support Vectors:  632

 ( 319 313 )


Number of Classes:  2

Levels:
 CH MM
```

Respecto a este nuevo modelo (kernel polinomial de rado 2) encontramos que el número de vectores de soporte (632) representa aproximadamente el 80% de las observaciones de los datos de entrenamiento.

En ese sentido, respecto al valor de costo especificado, podríamos inferir que aquel modelo no es tan apropiado (incluso menos apropiado que el modelo lineal en la parte b)) para clasificación de observaciones.

Parte g.c)

```r
tmp <- confusion_matrix_y_error(train, svmfit_g)
train_confusion_matrix_g.b <- tmp[["tabla"]]
train_confusion_matrix_g.b
```

```
    pred
true  CH  MM
  CH 487   0
  MM 292  21
```

```
train_error_g.b <- tmp[["error"]]
train_error_g.b
```

```
[1] 0.365
```

```
tmp <- confusion_matrix_y_error(test, svmfit_g)
test_confusion_matrix_g.b <- tmp[["tabla"]]
test_confusion_matrix_g.b
```

```
    pred
true  CH  MM
  CH 164   2
  MM 100   4
```

```
test_error_g.b <- tmp[["error"]]
test_error_g.b
```

```
[1] 0.3777778
```

Las tasas de error de entrenamiento y prueba son, respectivamente, 36.5 % y 37.7777778 % .

Parte g.d)

```
set.seed(4268)
tune.costo_g = tune(
  svm, Purchase ~., data = train,
  kernel = "polynomial", degree = 2,
  ranges = list(cost = seq(0.01, 10, length.out = 20))
)
summary(tune.costo_g)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 4.216316

- best performance: 0.1675

- Detailed performance results:
         cost   error dispersion
1   0.0100000 0.38750 0.06481812
2   0.5357895 0.20250 0.02024160
3   1.0615789 0.19000 0.02266912
4   1.5873684 0.18500 0.02751262
5   2.1131579 0.18000 0.03343734
6   2.6389474 0.17375 0.03143004
7   3.1647368 0.17000 0.02713137
8   3.6905263 0.16875 0.02841288
9   4.2163158 0.16750 0.03073181
10  4.7421053 0.16875 0.03131937
11  5.2678947 0.16875 0.03076005
12  5.7936842 0.16750 0.03129164
13  6.3194737 0.16750 0.03129164
14  6.8452632 0.17000 0.03129164
15  7.3710526 0.16875 0.03186887
16  7.8968421 0.16750 0.03238227
17  8.4226316 0.16750 0.02958040
18  8.9484211 0.16875 0.02841288
19  9.4742105 0.17000 0.02776389
20 10.0000000 0.17000 0.02776389
```

Respecto a los veinte casos evaluados para el valor de `costo`, el valor encontrado de **costo óptimo** es de `4.216316`.

Parte `g.e)`

```r
# Seleccionamos el mejor (según costo) modelo encontrado
svmfit_g.d <- tune.costo_g$best.model
svmfit_g.d
```

```
Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = seq(0.01,
    10, length.out = 20)), kernel = "polynomial", degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  4.216316
     degree:  2
     coef.0:  0

Number of Support Vectors:  371
```

```
tmp <- confusion_matrix_y_error(train, svmfit_g.d)
train_confusion_matrix_g.d <- tmp[["tabla"]]
train_confusion_matrix_g.d
```

```
    pred
true  CH  MM
  CH 452  35
  MM  82 231
```

```
train_error_g.d <- tmp[["error"]]
train_error_g.d
```

```
[1] 0.14625
```

```
tmp <- confusion_matrix_y_error(test, svmfit_g.d)
test_confusion_matrix_g.d <- tmp[["tabla"]]
test_confusion_matrix_g.d
```

```
    pred
true  CH  MM
  CH 147  19
  MM  28  76
```

```r
test_error_g.d <- tmp[["error"]]
test_error_g.d
```

```
[1] 0.1740741
```

Para el *mejor* modelo encontrado en la parte `d)`, las tasas de error de entrenamiento y prueba son, respectivamente, 14.625 % y 17.4074074 % .

Parte `h)`

Primero, presentamos en una tabla las tasas de error calculadas tanto en las partes `b, f, g` y sus subpartes.

En la siguiente tabla, respecto a la columna `modelo`, entiéndase aquellos nombres que terminan con `.b`, como referentes a los modelos creados en la parte `b` (o subparte `b`, para las partes `f` y `g`).

Análogamente para los nombres de esa columna que terminan en `.d`, haciendo referencia al *mejor modelo* encontrado, respecto al rango de `costo`, en los tres casos analizados para el kernel.

```r
resumen_modelos <- data.frame(
  modelo = c(
    "Lineal.b", "Lineal.d",
    "Radial.b", "Radial.d",
    "Polinomial.b", "Polinomial.d"
  ),
  train_error_rate = c(
    train_error_b, train_error_d,
    train_error_f.b, train_error_f.d,
    train_error_g.b, train_error_g.d
  ),
  test_error_rate = c(
    test_error_b, test_error_d,
    test_error_f.b, test_error_f.d,
    test_error_g.b, test_error_g.d
  )
)
knitr::kable(resumen_modelos)
```

modelo	train_error_rate	test_error_rate
Lineal.b	0.16750	0.1777778
Lineal.d	0.15750	0.1703704
Radial.b	0.39125	0.3851852
Radial.d	0.13750	0.1777778
Polinomial.b	0.36500	0.3777778
Polinomial.d	0.14625	0.1740741

Como ya es costumbre, compararemos a los seis modelos planteados según su **métrica de prueba**, en este caso, la **tasa de error de prueba**, no según la tasa de error de entrenamiento (métrica que puede ser sesgada a estar muy cercana a cero, en casos como overfitting).

Asimismo, recalcamos que los modelos cuyo nombre termina en `.d`, en comparación con los que acaban en `.b`, son referentes más apropiados respecto al tipo de modelo empleado. Esto debido a que su selección requirió el uso de **validación cruzada** (por medio de la función `tune()`), por lo cual se tiene una mayor certeza (estimación más precisa respecto a la población estadística) sobre su tasa de error de prueba asociado, comparado a los modelos de las (sub)partes `.b`, donde no se empleó validación cruzada.

En ese sentido, entre los modelos construidos, el enfoque que parece brindar mejores resultados (menor tasa de error de prueba) con estos datos es el modelo `Lineal.d`, con kernel lineal y costo de valor `8.948421`.

Sin embargo, debido a que los datos consisten de pocas observaciones (`OJ` tiene aproximadamente solo 1000 filas), y el hecho que el modelo `Polinomial.d` tiene una tasa de error de prueba relativamente pequeña, y **muy cercana** a la tasa de error de prueba del modelo `Lineal.d`; no se tiene suficiente evidencia como para declarar que el modelo `Lineal.d` es realmente más adecuado que el modelo `Polinomial.d` (kernel polinomial de grado 2).

Ambos enfoques/modelos mencionados presentan un relativamente buen enfoque para la clasificación de observaciones referentes al conjunto de datos `OJ`.

Problema 3

```
library(foreign)
```

```
departamentos <- read.spss(
  "./DepartamentosPeru.sav",
  use.value.labels = TRUE, max.value.labels = Inf,
  to.data.frame = TRUE
)
```

```
re-encoding from CP1252
```

```r
colnames(departamentos) <- tolower(colnames(departamentos))
head(departamentos)
```

```
   departamento  vida alfabetismo escolaridad  ingreso identidad     salud
1 AMAZONAS      72.40    88.03703    78.56076 204.6588  92.94668  9.654435
2 ANCASH        72.34    87.57830    86.30565 320.7716  97.04455 11.001835
3 APURÍMAC      71.77    78.32368    89.91364 203.3274  97.69712 12.320938
4 AREQUIPA      73.51    95.86742    90.73247 434.8148  97.98647 26.364593
5 AYACUCHO      70.92    82.19589    86.61866 206.8036  97.43440 11.722660
6 CAJAMARCA     72.07    82.85534    79.63548 215.6625  95.60393  6.953406
  saneamiento electrificacion   policia
1    38.98798        48.48029 0.7395820
2    59.88253        73.19061 0.8109211
3    46.32636        56.58622 1.1841460
4    75.18155        84.22968 1.9339311
5    44.05697        51.18760 0.5894502
6    52.04073        40.21862 0.5933728
```

```r
# Solo mantener columnas numéricas
nombres_departamentos <-  departamentos[,1]
departamentos <-  departamentos[,-1]
rownames(departamentos) <-  nombres_departamentos
head(departamentos)
```

```
              vida alfabetismo escolaridad  ingreso identidad     salud
AMAZONAS      72.40    88.03703    78.56076 204.6588  92.94668  9.654435
ANCASH        72.34    87.57830    86.30565 320.7716  97.04455 11.001835
APURÍMAC      71.77    78.32368    89.91364 203.3274  97.69712 12.320938
AREQUIPA      73.51    95.86742    90.73247 434.8148  97.98647 26.364593
AYACUCHO      70.92    82.19589    86.61866 206.8036  97.43440 11.722660
CAJAMARCA     72.07    82.85534    79.63548 215.6625  95.60393  6.953406
              saneamiento electrificacion   policia
AMAZONAS         38.98798        48.48029 0.7395820
ANCASH           59.88253        73.19061 0.8109211
APURÍMAC         46.32636        56.58622 1.1841460
AREQUIPA         75.18155        84.22968 1.9339311
AYACUCHO         44.05697        51.18760 0.5894502
CAJAMARCA        52.04073        40.21862 0.5933728
```

Parte a)

```
library(fpc)
```

```
Warning: package 'fpc' was built under R version 4.1.3
```

```
library(cluster)
```

```
Warning: package 'cluster' was built under R version 4.1.3
```

Criterio de Calinski-Harabasz

```
kmeansruns(scale(departamentos), criterion = "ch")
```

```
K-means clustering with 2 clusters of sizes 17, 8

Cluster means:
        vida alfabetismo escolaridad    ingreso  identidad      salud
1 -0.5603059  -0.4306735  -0.4150396 -0.4984779 -0.3405792 -0.5063146
2  1.1906500   0.9151812   0.8819592  1.0592655  0.7237308  1.0759186
  saneamiento electrificacion    policia
1  -0.5310237      -0.5179855 -0.4205665
2   1.1284254       1.1007192  0.8937038

Clustering vector:
AMAZONAS      ANCASH        APURÍMAC      AREQUIPA      AYACUCHO
            1             1             1             2             1
CAJAMARCA     CALLAO        CUSCO         HUANCAVELICA  HUÁNUCO
            1             2             1             1             1
ICA           JUNÍN         LA LIBERTAD   LAMBAYEQUE    LIMA
            2             1             1             2             2
LORETO        MADRE DE DIOS MOQUEGUA      PASCO         PIURA
            1             1             2             1             1
PUNO          SAN MARTÍN    TACNA         TUMBES        UCAYALI
            1             1             2             2             1

Within cluster sum of squares by cluster:
[1] 80.85455 27.84992
```

```
 (between_SS / total_SS =  49.7 %)

Available components:

 [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
 [6] "betweenss"    "size"         "iter"         "ifault"       "crit"
[11] "bestk"
```

Según el criterio de `Calinski-Harabasz`, el número adecuado de conglomerados que se debería usar para agrupar los departamentos es 2.

Asimismo, para este caso de dos conglomerados, un valor más cercano a `1` para `between_SS / total_SS` representa que la conglomeración es más adecuada.

Sin embargo, sgún el criterio de `Calinski-Harabaszs`, se ha obtenido `between_SS / total_SS = 49.7%`, valor no tan cercano a 1. Así que no basta este criterio (CH) para afirmar lo correcto que resulta agrupas por los departamentos en solo 2 conglomerados.

Criterio `anchura de silueta`

```r
kmeansruns(scale(departamentos), criterion = "asw")
```

```
K-means clustering with 2 clusters of sizes 8, 17

Cluster means:
        vida alfabetismo escolaridad    ingreso  identidad      salud
1  1.1906500   0.9151812   0.8819592  1.0592655  0.7237308  1.0759186
2 -0.5603059  -0.4306735  -0.4150396 -0.4984779 -0.3405792 -0.5063146
  saneamiento electrificacion    policia
1   1.1284254       1.1007192  0.8937038
2  -0.5310237      -0.5179855 -0.4205665

Clustering vector:
AMAZONAS      ANCASH        APURÍMAC      AREQUIPA      AYACUCHO
            2             2             2             1             2
CAJAMARCA     CALLAO        CUSCO         HUANCAVELICA  HUÁNUCO
            2             1             2             2             2
ICA           JUNÍN         LA LIBERTAD   LAMBAYEQUE    LIMA
            1             2             2             1             1
LORETO        MADRE DE DIOS MOQUEGUA      PASCO         PIURA
            2             2             1             2             2
PUNO          SAN MARTÍN    TACNA         TUMBES        UCAYALI
```

```
 2  2  1  1  2

Within cluster sum of squares by cluster:
[1] 27.84992 80.85455
 (between_SS / total_SS =  49.7 %)

Available components:

 [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
 [6] "betweenss"    "size"         "iter"         "ifault"       "crit"
[11] "bestk"
```

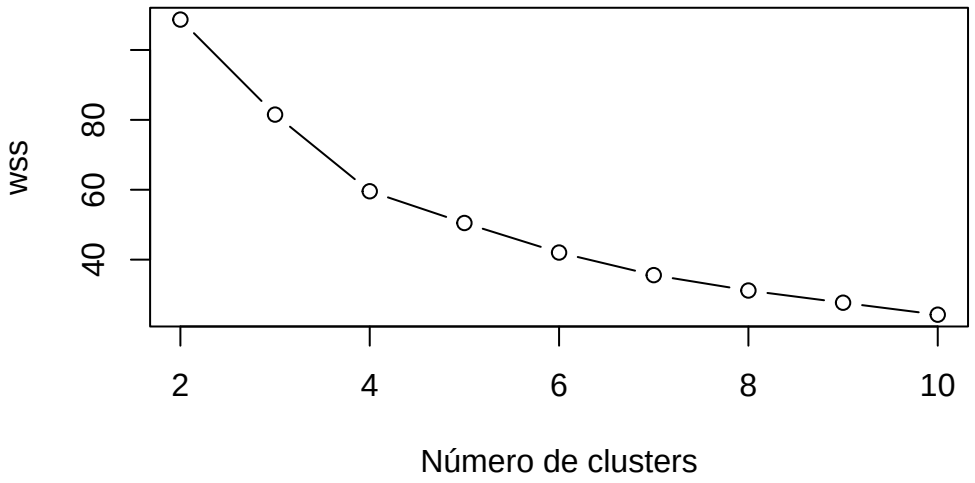
Según el criterio de `anchura de silueta`, el número adecuado de conglomerados que se debería usar para agrupar los departamentos es 2.

Similar al caso del criterio previo, se obtiene `between_SS / total_SS = 49.7%`, por lo que concluimos lo mismo que en el criterio anterior.

Criterio de `mean individual silhoette widths`

```r
min_num_clusters <- 2
max_num_clusters <- 10

# Suma de cuadrados dentro de clusters
wss <- numeric()
for (h in min_num_clusters:max_num_clusters) {
  b <- kmeans(
    scale(departamentos), h,
    # Argumento importante para que los elementos de wss decrezcan
    nstart = 50
  )
  wss <- append(wss, b$tot.withinss)
}
plot(
  min_num_clusters:max_num_clusters, wss, type = "b",
  xlab = "Número de clusters"
)
```




```
mean_ind_sil_widths <- numeric()
diss.departamentos <- daisy(scale(departamentos))

for (h in min_num_clusters:max_num_clusters) {
  res <- kmeans(scale(departamentos), h)
  resumen <- summary(silhouette(res$cluster, diss.departamentos))
  # Extraer la media
  mean_ind_sil_widths <- append(mean_ind_sil_widths, unname(resumen[[1]]["Mean"]))
}
mean_ind_sil_widths
```

```
[1] 0.3948749 0.2767644 0.2623672 0.3075457 0.2584330 0.1928748 0.2053959
[8] 0.2162199 0.2137904
```

La silueta representa qué tan bien está agrupada una observación en su conglomerado respectivo. A **mayor silueta**, mayor similitud de la observación a las observaciones en su conglomerado.

```
# Cantidad de clusters, con el cual se maximiza el promedio
# de los anchos de siluetas de las observaciones
(min_num_clusters:max_num_clusters)[which.max(mean_ind_sil_widths)]
```

```
[1] 2
```

Por lo tanto, según el criterio de `promedio de anchos individuales de silueta`, el número adecuado de conglomerados que se debería usar para agrupar los departamentos es 2.

Parte `b)`