Ciencia de Datos para Políticas Públicas

Clase 08 - Árboles de decisión y 'clustering'

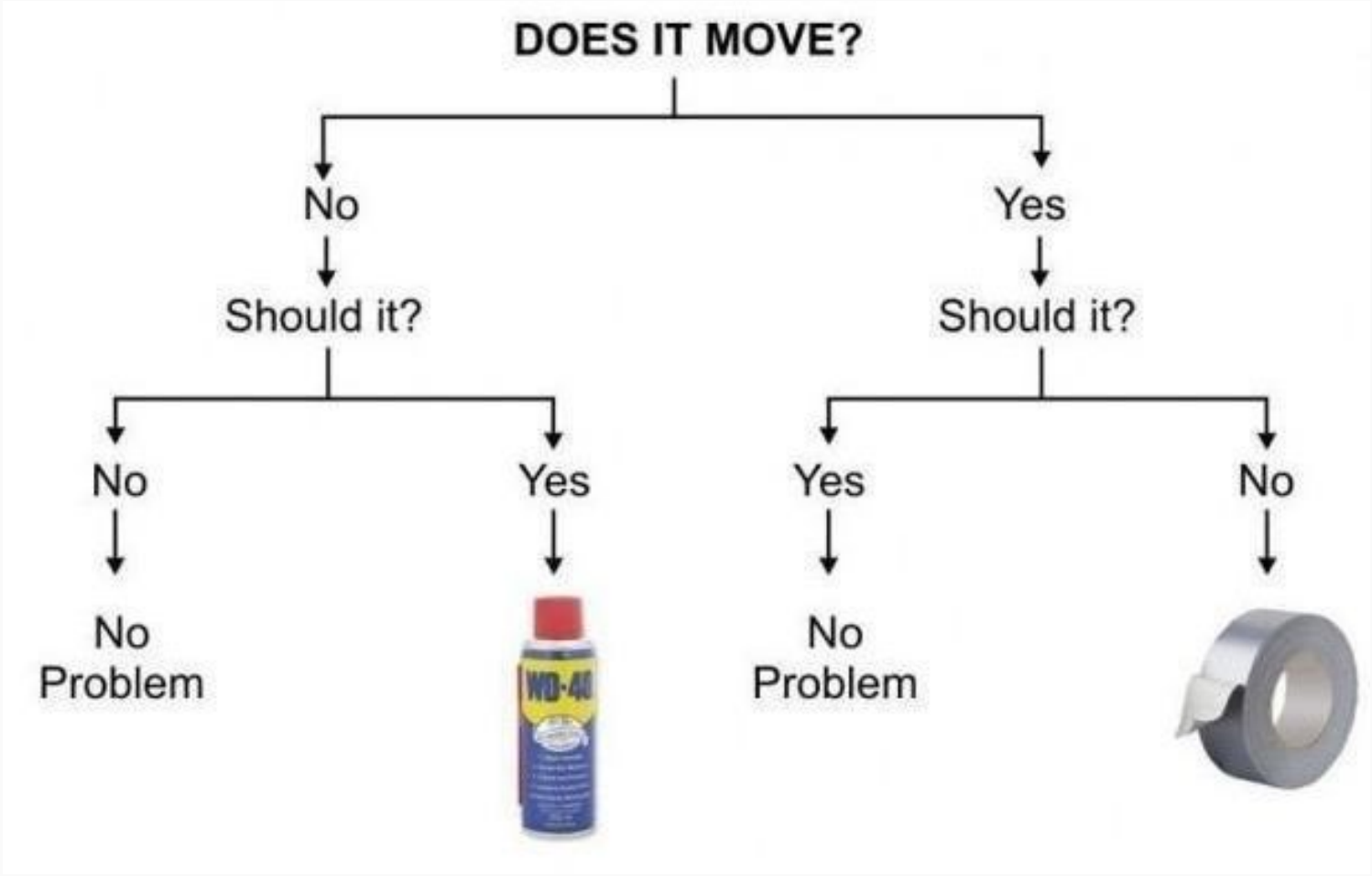
Pablo Aguirre Hormann
30/09/2020

¿Qué veremos hoy?

- Árboles de decisión
- Clustering
    - *k-means*

Árboles de decisión

Los han visto antes

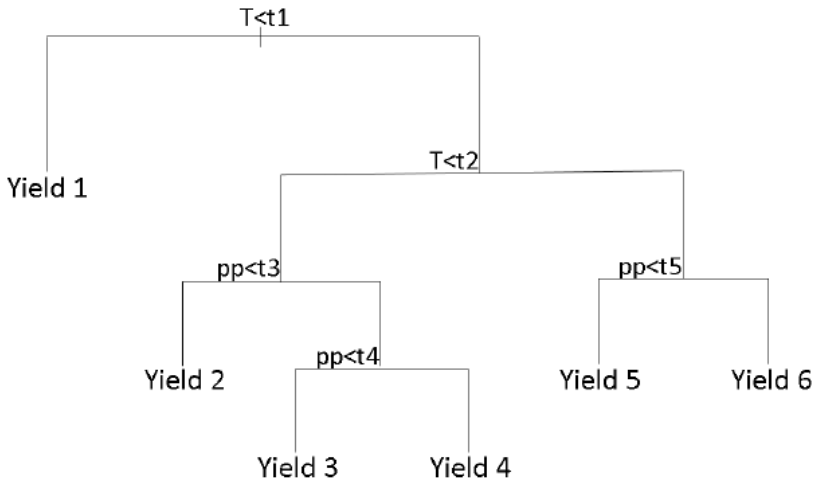


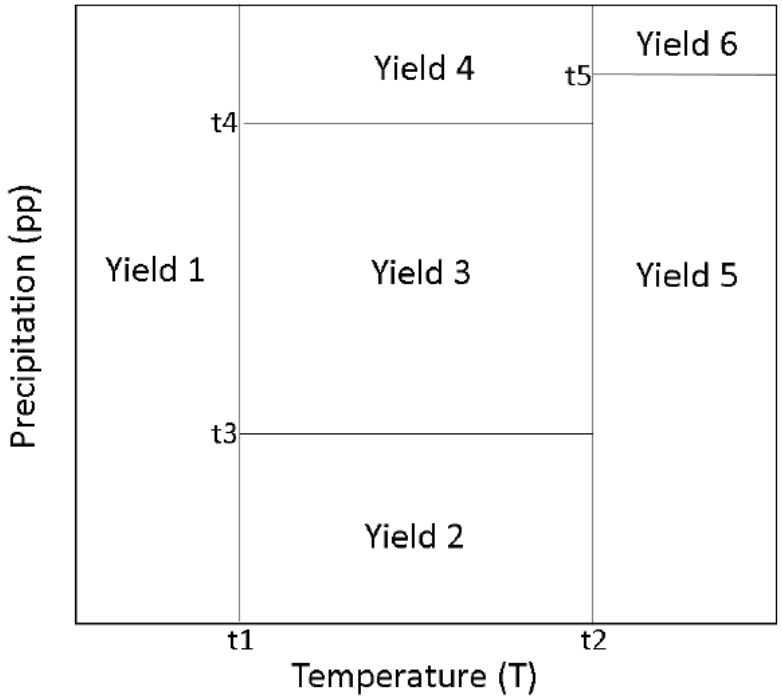
¿Cómo funcionan?

- Existen muchas metodologías para generar árboles
    - La más diseminada se conoce como el algoritmo de CART (Breiman, 1984)
- Separar los datos en subgrupos y asignar una constante a cada observación dentro de estos
- Los subgrupos se generan a través de divisiones binarias sucesivas (división recursiva) en base a las variables disponibles
- La constante a asignar corresponde al promedio de los valores dentro de cada subgrupo

Se pueden usar tanto para regresión como para clasificación

Descripción gráfica





Terminología

- Cada resultado (*Yield X*) es un **nodo terminal u hoja**
- Cada división es un **nodo interno**
- Los segmentos que conectan cada nodo son las **ramas**

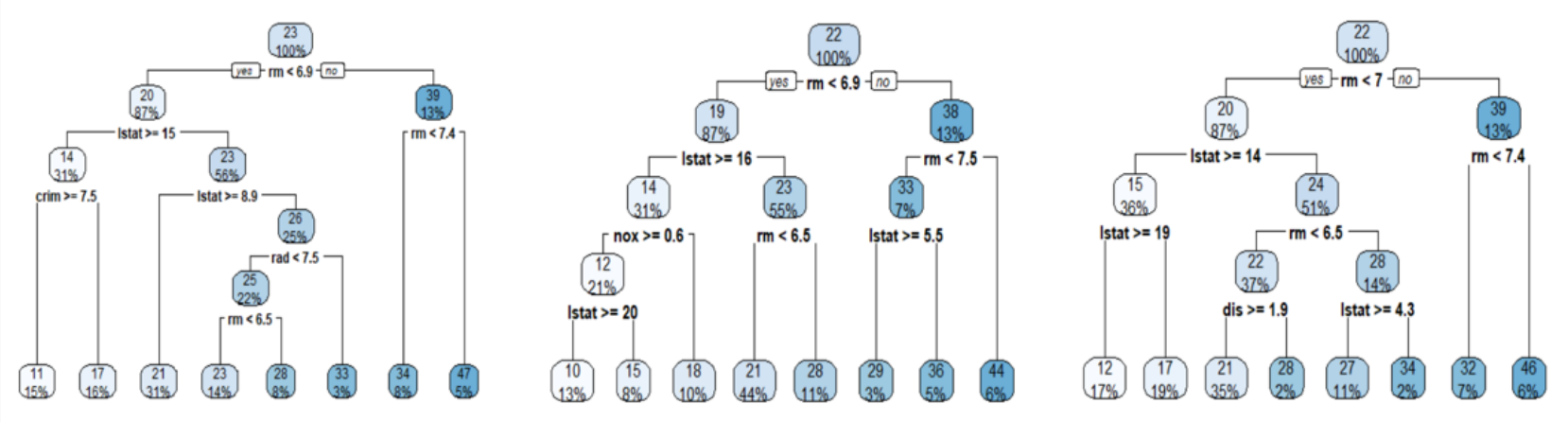
¿Cómo definir los subgrupos/áreas?

- Las divisiones se realizan de forma *top-down*
    - Una división generada no cambiará según las siguientes divisiones
- Se comienza con todos los datos y se busca el valor que permite separar en R_1 y R_2 buscando minimizar la suma de cuadrados residuales

$$min\{SCE = \sum_{i \in R_1}(y_i - c_1)^2 + \sum_{i \in R_2}(y_i - c_2)^2\}$$

- Se repite el procedimiento en cada una de las regiones generadas y así sucesivamente hasta que se cumpla algún criterio de término
- Si no se restringe, el algoritmo genera árboles muy complejos que tenderán a sobreajustar los datos (buen ajuste dentro de muestra pero malas predicción fuera de muestra).

Ejemplos



- Las divisiones son similares en la parte superior pero comienzan a diferenciarse (bastante) al ir bajando hasta las hojas
- Los nodos "más profundos" tienden a sobreajustar los datos
  - Cambios en las muestras generan resultados muy variables en las estimaciones/predicciones
- Los árboles se pueden "podar" para mejorar la predicción (introducir sesgo para mejorar predicción)

Criterio de costo de complejidad

- Para encontrar el mejor árbol de decisión (en términos predictivos), lo que se puede hacer es generar un primer árbol muy grande o complejo y luego podarlo hasta encontrar el punto óptimo de complejidad.
- Esto lo podemos implementar usando un parámetro de *costo de complejidad* (α) penalizando el número de nodos (T) en en un árbol.

$$min\{SCE + \alpha|T|\}$$

- Para un valor de α encontramos el árbol podado más pequeño con el error penalizado más bajo.
- La penalización debería recordarles algo de lo visto con **LASSO**
  - menores valores de α producen modelos más complejos (árboles más grandes)
  - mayores valores de α producen modelos más simples (árboles más pequeños)
- ¿Cómo encontrar α? $\rightarrow$ **cross-validation**

Ventajas y desventajas

Ventajas

- Alta interpretabilidad
- Fácil entender que variables son más importantes
- Rápido para realizar predicciones (algoritmo no es complejo)
- Puede adaptarse a problemas no lineales

Desventajas

- Árboles individuales presentan alta varianza
- Por ende, no suelen tener buen rendimiento de predicción

Existen formas de usar árboles y lidiar con las desventajas

Árboles de decisión en R

```
library(tidyverse) # manejo de datos
library(rpart) # árboles de decisión
library(rpart.plot) # gráficos de árboles de decisión
library(AmesHousing) # datos
library(rsample) # construcción de train/test

datos_casas ← make_ames()
dim(datos_casas)
```

```
## [1] 2930   81
```

```
set.seed(123)
split ← initial_split(datos_casas , prop = .7)
datos_train ← training(split)
datos_test  ← testing(split)

dim(datos_train)
```

```
## [1] 2051   81
```

```
dim(datos_test)
```

Árboles de decisión en R (cont)

```
m1 ← rpart(
  formula = Sale_Price ~ .,
  data    = datos_train,
  method  = "anova") # "class" para clasificación

str(m1)
```

```
## List of 15
##  $ frame              :'data.frame':    25 obs. of  8 variables:
##   ..$ var       : chr [1:25] "Overall_Qual" "Neighborhood" "First_Flr_SF" "Overall_Qual"
##   ..$ n         : int [1:25] 2051 1703 1015 611 152 459 404 359 45 688 ...
##   ..$ wt        : num [1:25] 2051 1703 1015 611 152 ...
##   ..$ dev       : num [1:25] 1.27e+13 4.03e+12 1.36e+12 4.92e+11 1.05e+11 ...
##   ..$ yval      : num [1:25] 180776 156431 131803 118301 91653 ...
##   ..$ complexity: num [1:25] 0.4669 0.1196 0.022 0.0113 0.01 ...
##   ..$ ncompete  : int [1:25] 4 4 4 4 0 0 4 0 0 4 ...
##   ..$ nsurrogate: int [1:25] 5 5 5 5 0 0 5 0 0 5 ...
##  $ where              : Named int [1:2051] 8 8 9 12 12 17 12 12 12 17 ...
##   ..- attr(*, "names")= chr [1:2051] "1" "2" "3" "4" ...
##  $ call               : language rpart(formula = Sale_Price ~ ., data = datos_train, met
##  $ terms              :Classes 'terms', 'formula'  language Sale_Price ~ MS_SubClass + M
##   .. ..- attr(*, "variables")= language list(Sale_Price, MS_SubClass, MS_Zoning, Lot_Fro
##   .. ..- attr(*, "factors")= int [1:81, 1:80] 0 1 0 0 0 0 0 0 0 0 ...
```

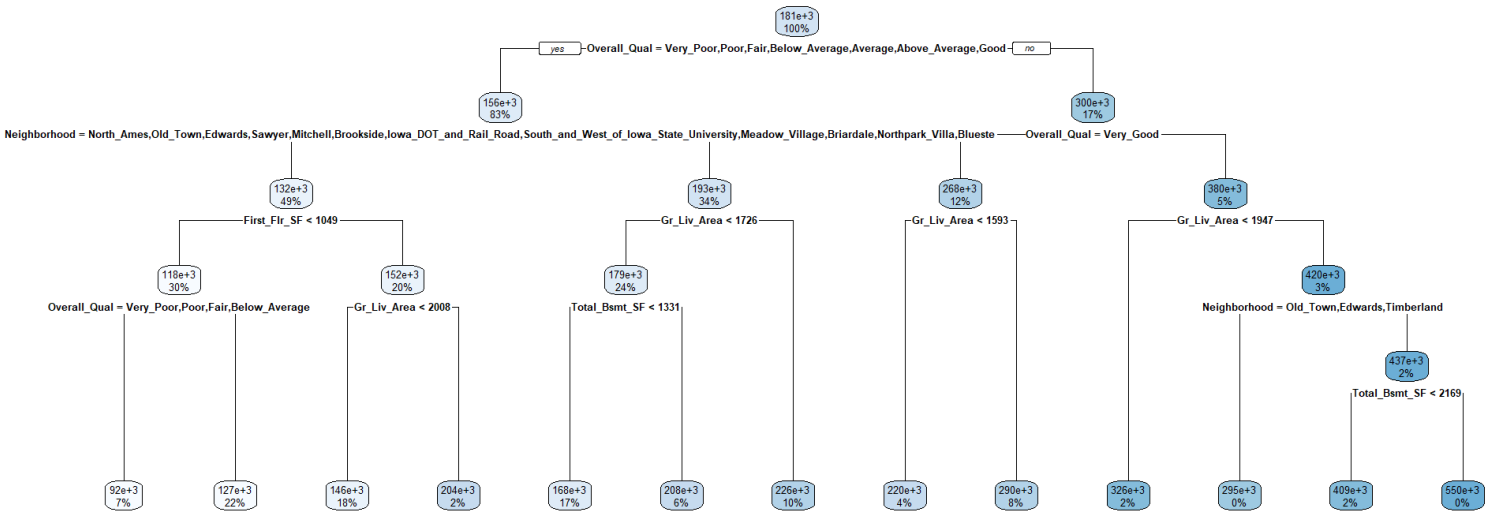
Árboles de decisión en R (cont)

```
m1
```

```
## n= 2051
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 2051 1.273987e+13 180775.50
##    2) Overall_Qual=Very_Poor,Poor,Fair,Below_Average,Average,Above_Average,Good 1703 4.0
##      4) Neighborhood=North_Ames,Old_Town,Edwards,Sawyer,Mitchell,Brookside,Iowa_DOT_and_
##        8) First_Flr_SF< 1048.5 611 4.924281e+11 118301.50
##          16) Overall_Qual=Very_Poor,Poor,Fair,Below_Average 152 1.053743e+11  91652.57 *
##          17) Overall_Qual=Average,Above_Average,Good 459 2.433622e+11 127126.40 *
##        9) First_Flr_SF≥1048.5 404 5.880574e+11 152223.50
##          18) Gr_Liv_Area< 2007.5 359 2.957141e+11 145749.50 *
##          19) Gr_Liv_Area≥2007.5 45 1.572566e+11 203871.90 *
##      5) Neighborhood=College_Creek,Somerset,Northridge_Heights,Gilbert,Northwest_Ames,Sa
##       10) Gr_Liv_Area< 1725.5 482 5.162415e+11 178531.00
##         20) Total_Bsmt_SF< 1331 352 2.315412e+11 167759.00 *
##         21) Total_Bsmt_SF≥1331 130 1.332603e+11 207698.30 *
##       11) Gr_Liv_Area≥1725.5 206 3.056877e+11 226068.80 *
##    3) Overall_Qual=Very_Good,Excellent,Very_Excellent 348 2.759339e+12 299907.90
##      6) Overall_Qual=Very_Good 249 9.159879e+11 268089.10
```

Árboles de decisión en R (cont)

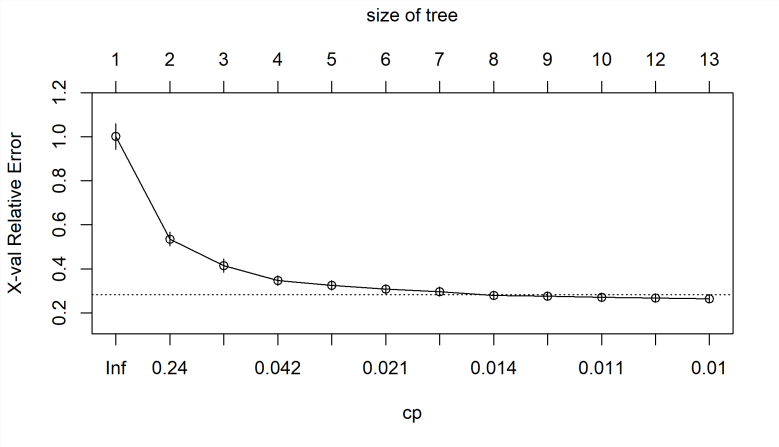
```
rpart.plot(m1)
```



Árboles de decisión en R (cont)

- `rpart` está automáticamente aplicando valores de costo de complejidad, α, para podar el árbol
  - por defecto usa *10-fold cross validation*

```
plotcp(m1)
```



- El eje y corresponde al error de predicción obtenido por *cross validation*
- El eje x inferior (**cp**) corresponde al costo de complejidad
- El eje x superior corresponde al número de nodos terminales u hojas

Árboles de decisión en R (cont)

```
m1$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1  0.46690132      0 1.0000000 1.0009222 0.05855161
## 2  0.11961409      1 0.5330987 0.5347929 0.03116217
## 3  0.06955813      2 0.4134846 0.4151417 0.03058554
## 4  0.02559992      3 0.3439265 0.3461258 0.02207839
## 5  0.02196620      4 0.3183265 0.3242197 0.02182111
## 6  0.02023390      5 0.2963603 0.3074877 0.02129292
## 7  0.01674138      6 0.2761264 0.2963372 0.02106996
## 8  0.01188709      7 0.2593850 0.2795199 0.01903482
## 9  0.01127889      8 0.2474980 0.2762666 0.01936472
## 10 0.01109955      9 0.2362191 0.2699895 0.01902217
## 11 0.01060346     11 0.2140200 0.2672133 0.01883219
## 12 0.01000000     12 0.2034165 0.2635207 0.01881691
```

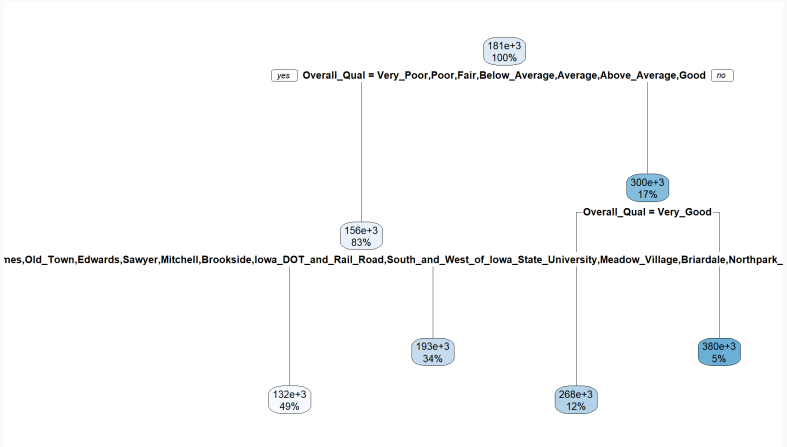
Árboles de decisión en R (cont)

Además del costo de complejidad, α, otros hiperparámetros a ajustar son:

- `minsplit`: número mínimo de observaciones requeridas para realizar una división. Por defecto es 20.
- `maxdepth`: número máximo de nodos internos entre el nodo inicial y las hojas. Por defecto es 30.

Con `rpart` podemos usar el argumento `control` para indicar valores de hiperparámetros.

```
m3 ← rpart(
    formula = Sale_Price ~ .,
    data    = datos_train,
    method  = "anova",
    control = list(minsplit = 10,
                   maxdepth = 2))
```



Árboles de decisión en R (cont)

Para ver la combinación ideal de hiperparámetros podemos generar una "grilla de búsqueda".

```
hiper_grid ← expand.grid(
  minsplit = seq(5, 20, 1),
  maxdepth = seq(8, 15, 1)
)

head(hiper_grid)
```

```
##   minsplit maxdepth
## 1        5        8
## 2        6        8
## 3        7        8
## 4        8        8
## 5        9        8
## 6       10        8
```

```
nrow(hiper_grid)
```

```
## [1] 128
```

Podemos luego implementar un *loop* para entrenar los modelos usando las distintas combinaciones de la grilla (`hiper_grid`).

```
modelos ← list()

for (i in 1:nrow(hiper_grid)) {

  # definir hiperparametros de la iterac
  minsplit ← hiper_grid$minsplit[i]
  maxdepth ← hiper_grid$maxdepth[i]

  # entrenar modelos y guardar en la lis
  modelos[[i]] ← rpart(
    formula = Sale_Price ~ .,
    data    = datos_train,
    method  = "anova",
    control = list(minsplit = minsplit,
                   maxdepth = maxdepth)
    )
}
```

Árboles de decisión en R (cont)

Ahora generaremos una función para extraer el valor óptimo de α para cada modelo generado y su correspondiente error.

```
# función para obtener el valor óptimo c
obtener_cp ← function(x) {
  min    ← which.min(x$cptable[, "xerrc
  cp ← x$cptable[min, "CP"]
}

# función para obtener el valor de erro
obtener_min_error ← function(x) {
  min    ← which.min(x$cptable[, "xerrc
  xerror ← x$cptable[min, "xerror"]
}
```

```
(mejores_valores ← hiper_grid %>%
  mutate(
    cp    = map_dbl(modelos, obtener_cp)
    error = map_dbl(modelos, obtener_mir
    ) %>%
  arrange(error) %>%
  slice(1))
```

```
##   minsplit maxdepth          cp     error
## 1       17       12 0.01060346 0.2628987
```

Árboles de decisión en R (cont)

```
arbol_optimo ← rpart(
    formula = Sale_Price ~ .,
    data    = datos_train,
    method  = "anova",
    control = list(minsplit = mejores_valores$minsplit,
                   maxdepth = mejores_valores$maxdepth,
                   cp = 0.01)
    )

pred ← predict(arbol_optimo, newdata = datos_test)
sqrt(mean((pred - datos_test$Sale_Price)^2))
```

```
## [1] 39852.01
```

```
summary(datos_casas$Sale_Price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   12789  129500  160000  180796  213500  755000
```

Extensiones de los árboles de decisión

- Árboles individuales suelen tener alta varianza (problema a la hora de predecir)
- Extensiones posibles:
  - ***Bagging***: promediar distintos árboles individuales que son entrenados en subconjuntos de las observaciones generados por *bootstrap*
  - ***Random Forest***: además de realizar muestras por *bootstrap* los distintos árboles se estiman con subconjuntos aleatorios de las variables disponibles
  - ***Boosting***: se "ensamblan" secuencialmente distintos árboles pequeños que van aprendiendo del árbol anterior

Clustering

¿Qué es el *clustering*?

- Es un tipo de análisis que divide las observaciones en grupos que comparten características similares
- La idea es la siguiente, generar grupos donde:
  - las observaciones dentro de un grupo sean similares
  - los grupos sean lo más distintos posibles
- Existen distintos tipos de *clustering*
  - **k-means**
  - k-medoids (PAM)
  - jerárquico
  - basado en modelo
  - basado en densidad
  - y más

¿Cómo funciona?

- Los algoritmos de *clustering* usan alguna medida de distancia entre las observaciones para asignar a grupos
- Las medidas de distancia más usadas son:

Distancia Euclidiana

$$d_{euc}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Distancia Manhattan

$$d_{man}(x,y) = \sum_{i=1}^{n}|(x_i - y_i)|$$

- Otros tipos de medidas de distancia:
  - Pearson
  - Spearman
  - Kendall

k-means clustering

- *k-means* es el de los algoritmos no supervisados más utilizados
- Permite separar un *data set* en k grupos (o *clusters*)
    - k se define *a priori*

Idea general

- Definir *clusters* que permitan minimizar la variación total ***intra-cluster***
- Existen distintos algoritmos de *k-means*
    - El más utilizado es el algoritmo de Hartigan-Wong (1979)

$$W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

donde x_i es un dato perteneciente al *cluster* C_k y μ_k es el promedio de los puntos asignados al *cluster* C_k.

k-means clustering (cont)

Cada observación, x_i, se asigna a un *cluster* con el objetivo de minimizar la suma del cuadrado de la distancia entre las observaciones y el centro de su *cluster*, μ_k

$$min \sum_{k=1}^{k} W(C_k) = \sum_{k=1}^{k} \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

Algoritmo *k-means*

1. Definir el número de *cluster*, k, a construir
2. Seleccionar aleatoriamente k observaciones como los centros iniciales de cada *cluster*
3. Asignar cada observación a su centro más cercano (según distancia Euclidiana)
4. Para cada *cluster*, modificar el centro calculando el nuevo promedio según los datos asignados
5. Iteratívamente minimizar la varianza dentro de los *cluster*. Osea, repetir los pasos **3** y **4** hasta que los *cluster* no cambien o bien hasta alcanzar el máximo de iteraciones definido (10 en el caso de `R`).

k-means en R

```
summary(USArrests)
```

```
##      Murder          Assault         UrbanPop          Rape
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

```
df ← USArrests %>%
  mutate(across(where(is.numeric), scale))

rownames(df) ← rownames(USArrests)

summary(df)
```

```
##       Murder.V1           Assault.V1          UrbanPop.V1
##  Min.   :-1.6044046   Min.   :-1.5090416   Min.   :-2.3171363
##  1st Qu.:-0.8524835   1st Qu.:-0.7410815   1st Qu.:-0.7627068
##  Median :-0.1235217   Median :-0.1411127   Median : 0.0317794
##  Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
##  3rd Qu.: 0.7948553   3rd Qu.: 0.9388312   3rd Qu.: 0.8435371
```

k-means en R (cont)

```
kmean2 ← kmeans(df, centers = 2)
str(kmean2)
```

```
## List of 9
##  $ cluster     : Named int [1:50] 1 1 1 2 1 1 2 2 1 1 ...
##   ..- attr(*, "names")= chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ centers     : num [1:2, 1:4] 1.005 -0.67 1.014 -0.676 0.198 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
##  $ totss       : num 196
##  $ withinss    : num [1:2] 46.7 56.1
##  $ tot.withinss: num 103
##  $ betweenss   : num 93.1
##  $ size        : int [1:2] 20 30
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

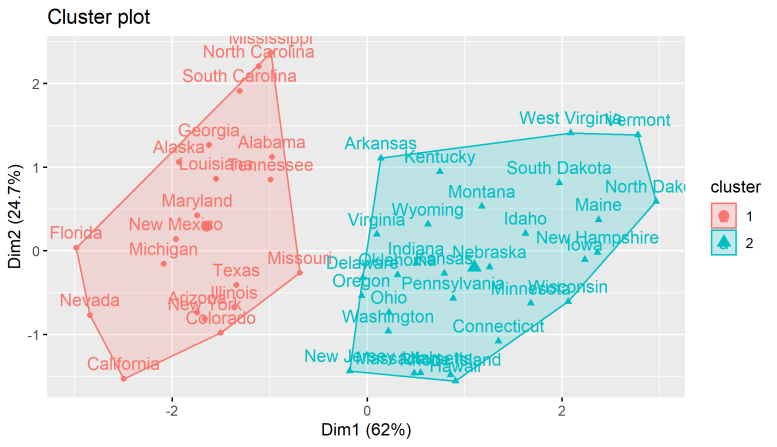
k-means en R (cont)

```
kmean2
```

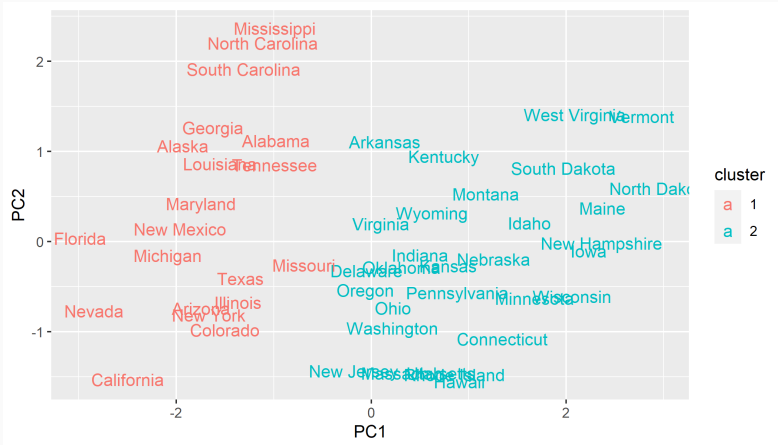
```
## K-means clustering with 2 clusters of sizes 20, 30
##
## Cluster means:
##      Murder    Assault   UrbanPop       Rape
## 1  1.004934  1.0138274  0.1975853  0.8469650
## 2 -0.669956 -0.6758849 -0.1317235 -0.5646433
##
## Clustering vector:
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              1              2              2              1              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              2              2              1              2              2
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              2              2              1              2              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              2              1              1
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              2              2              1              2              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
```

k-means en R (cont)

```
library(factoextra)
fviz_cluster(kmean2, data = df)
```

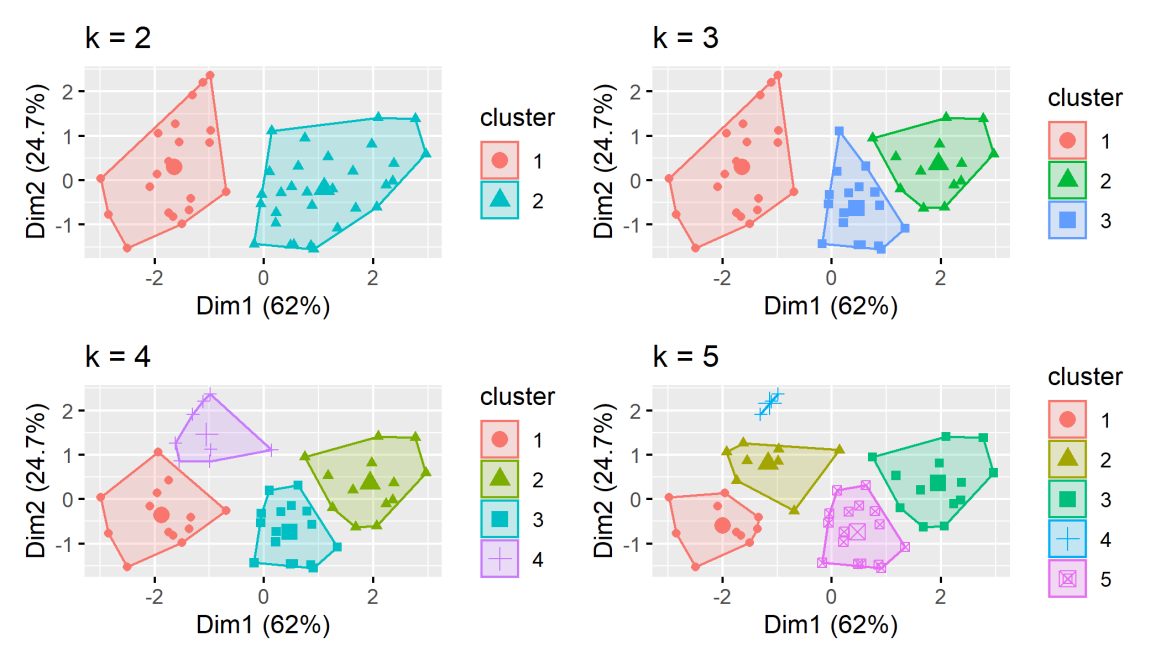



```
df_pca ← prcomp(df)

df %>%
  bind_cols(as_tibble(df_pca$x)) %>%
  mutate(cluster = as.factor(kmean2$clus
         estado = row.names(.)) %>%
  ggplot(aes(PC1, PC2, color = cluster,
  geom_text()
```



k-means en R (cont)

```
kmean3 ← kmeans(df, centers = 3)
kmean4 ← kmeans(df, centers = 4)
kmean5 ← kmeans(df, centers = 5)

p1 ← fviz_cluster(kmean2, geom = "point", data = df) + ggtitle("k = 2")
p2 ← fviz_cluster(kmean3, geom = "point",  data = df) + ggtitle("k = 3")
p3 ← fviz_cluster(kmean4, geom = "point",  data = df) + ggtitle("k = 4")
p4 ← fviz_cluster(kmean5, geom = "point",  data = df) + ggtitle("k = 5")
```

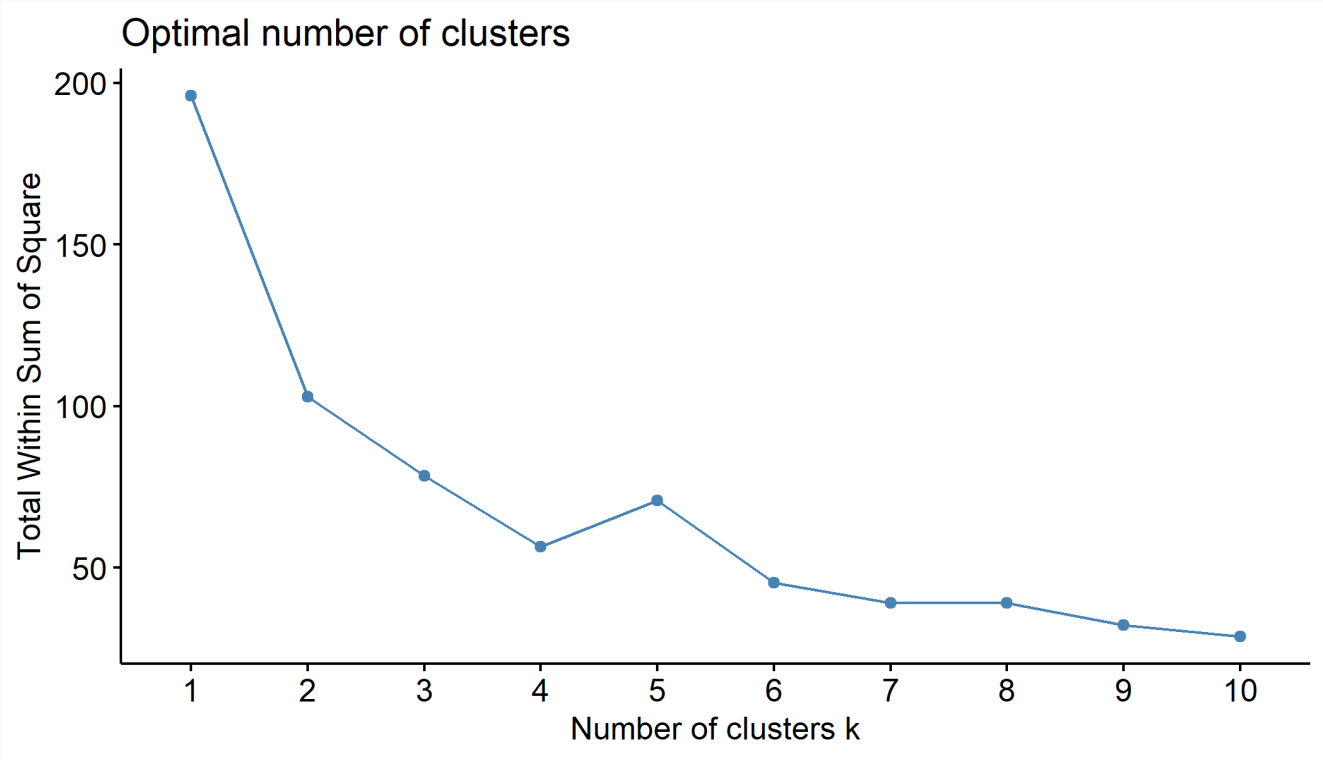


k-means en R (cont)

¿Cómo decidir k?

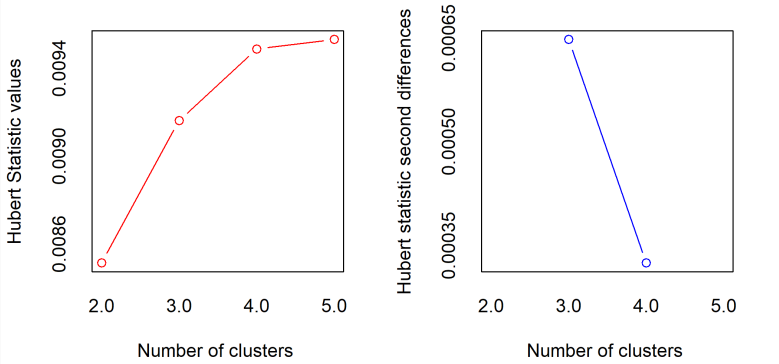
Existen distintas métricas (ej, *elbow*, *silhouette*, *gap*, otros)

```
fviz_nbclust(df, kmeans, method = "wss")
```



k-means en R (cont)

```
library(NbClust)

nbclust_out ← NbClust(
  data = df,
  distance = "euclidean",
  min.nc = 2,
  max.nc = 5,
  method = "kmeans"
)
```

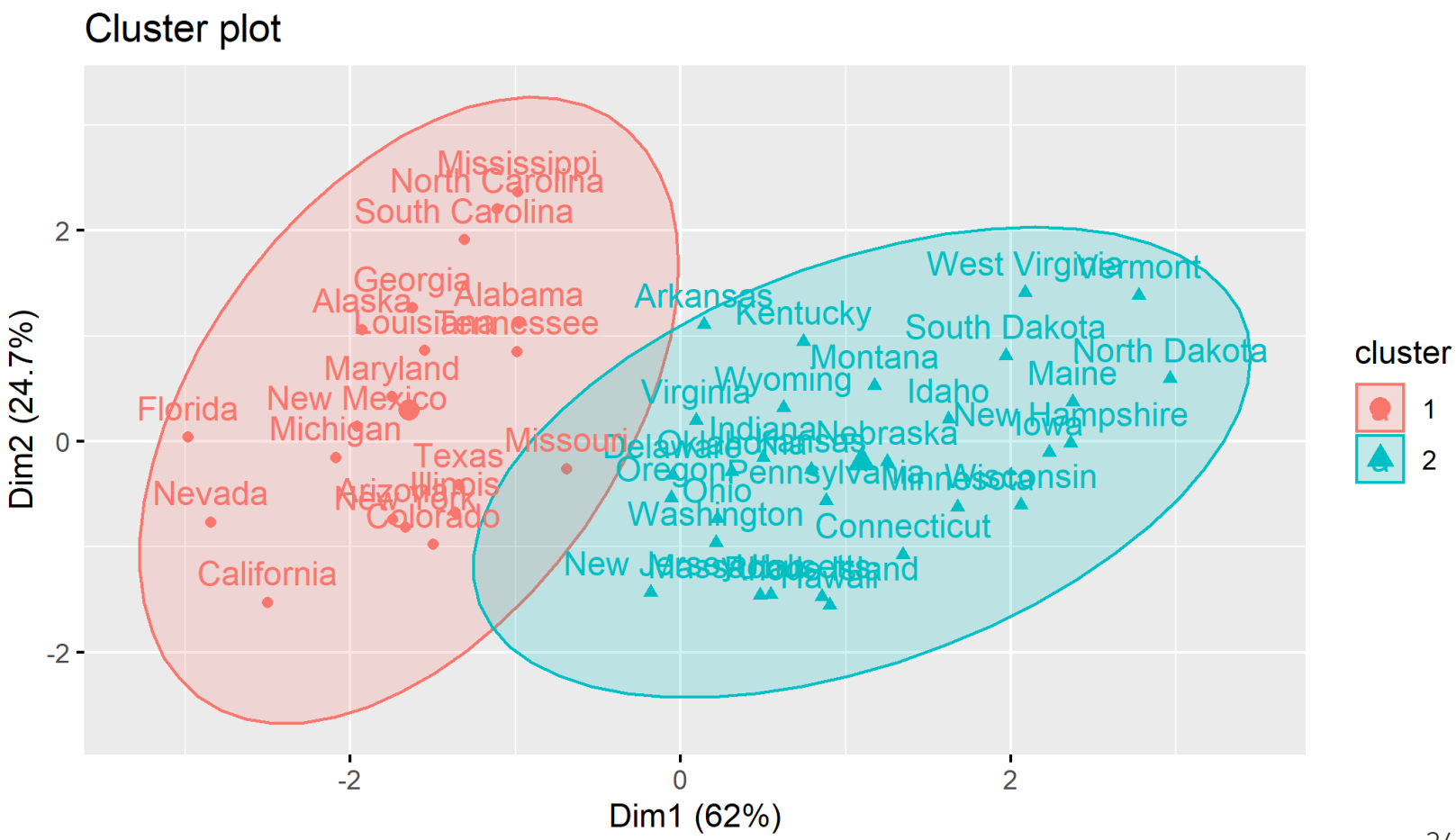



```
nbclust_out$Best.nc %>%
  as.tibble() %>%
  slice(1) %>%
  pivot_longer(1:26) %>%
  count(value)
```

```
## # A tibble: 5 x 2
##   value     n
##   <dbl> <int>
## 1     0     2
## 2     2    13
## 3     3     7
## 4     4     2
## 5     5     2
```

k-means en R

```
km_res ← kmeans(df, centers = 2, nstart = 20)
fviz_cluster(km_res, df, ellipse.type = "norm")
```



Desventajas de *k-means*

- Especificación previa de k
  - por ejemplo, *clustering* jerárquico no lo requiere
- Sensible a *outliers*
  - existe una alternativa (PAM) que es menos sensible
- Para relaciones/interacciones menos complejas existen algoritmos más eficientes
  - t-sne, autoencoders, etc
  - también son más costosos computacionalmente

Siguiente clase

- Última clase de contenidos
- *Web scraping*
- Casos de ML para políticas públicas

Sobre el trabajo

- Cambio de fecha para el trabajo final
  - Sábado después de la presentación
- Presentación
  - 5-8 minutos
  - 2-3 preguntas máximo