FUNDAMENTOS DE INGENIERÍA DE DATOS

MEMORIA TRABAJO PRÁCTICO

GRUPO 9

Máster en Ingeniería del Software: Datos, Cloud y Gestión TI

Universidad de Sevilla

Análisis supervisado, Análisis No supervisado & BIG ML: Clusters

Blanco Ferreira, Juan MIguel

Bravo Llanos, Alfonso

García Pascual, César

Guerrero Díaz, Alejandro

Repositorio: https://github.com/cesgarpas/fid-9

Curso 2020-21

0. Reparto de horas

Las horas mostradas a continuación aplican a todos los miembros del grupo por igual, ya que la totalidad de la realización de este proyecto tuvo lugar de forma colaborativa mediante llamadas de voz, participando en el 100% de los casos el equipo al completo. Hemos creído conveniente esta forma de trabajar, en lugar de dividirnos el trabajo y realizarlo por separado por dos motivos principalmente: por un lado hemos conseguido una mayor aportación de ideas, rapidez a la hora de solucionar problemas o errores y conclusiones comunes, y por otro lado hemos evitado que un subgrupo domine la parte que ellos han hecho, pero desconozca por completo el contenido y las tareas de las secciones de sus otros compañeros. Al participar todos en todo, el aprendizaje de los cuatro componentes del grupo ha sido mucho mayor, más rápido y más rico.

Tarea	Nº horas
Selección datasets	2
Preprocesamiento Knime (supervisado)	5,5
Preprocesamiento R (supervisado)	2,5
Decisión algoritmos (supervisado)	1,5
Aplicación Rpart	2
Aplicación Naive Bayes	1,5
Aplicación Ctree2	1
Preprocesamiento R (no supervisado)	0,5
Aplicación Kmeans	1
Aplicación Kmedioids	1
Aplicación Agnes	1,5
BigML	3
Realización memoria (preprocesamiento)	1,5
Realización memoria (supervisado)	2
Realización memoria (no supervisado)	1,5
Realización memoria (BigML)	1,5
Realización presentación	3
Preparación presentación	1

Horas por miembro:
33,5
Horas totales del Equipo:
134

1. Introducción

El objetivo del trabajo práctico de la asignatura actual consiste en el análisis de conjuntos de datos mediante el uso de técnicas de aprendizaje supervisado y no supervisado, así como el estudio de la herramienta Big ML.

Durante la realización de este proyecto, se han usado diferentes técnicas y algoritmos, que se explicarán a lo largo del documento, para realizar los análisis basados en clasificación y regresión, con los que sacar conclusiones a través del estudio de los resultados obtenidos.

2. Datasets Utilizados

En primer lugar, para el análisis supervisado hemos utilizado un dataset de ventas de videojuegos que está formado por 11 columnas y con un total de 16600 entradas de datos.

	A	B	C	D	E	F	G	H	I	J	K
1	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sa
2	1	Wii Sports	Wii	2006	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
3	2	Super Mario Bros.	NES	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
4	3	Mario Kart Wii	Wii	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
5	4	Wii Sports Resort	Wii	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	
6	5	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
7	6	Tetris	GB	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
8	7	New Super Mario Bros.	DS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
9	8	Wii Play	Wii	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02
10	9	New Super Mario Bros. Wii	Wii	2009	Platform	Nintendo	14.59	7.06	4.7	2.26	28.62
11	10	Duck Hunt	NES	1984	Shooter	Nintendo	26.93	0.63	0.28	0.47	28.31
12	11	Nintendogs	DS	2005	Simulation	Nintendo	9.07	11	1.93	2.75	24.76
13	12	Mario Kart DS	DS	2005	Racing	Nintendo	9.81	7.57	4.13	1.92	23.42
14	13	Pokemon Gold/Pokemon Silver	GB	1999	Role-Playing	Nintendo	9	6.18	7.2	0.71	23.1
15	14	Wii Fit	Wii	2007	Sports	Nintendo	8.94	8.03	3.6	2.15	22.72
16	15	Wii Fit Plus	Wii	2009	Sports	Nintendo	9.09	8.59	2.53	1.79	
17	16	Kinect Adventures!	X360	2010	Misc	Microsoft Ga	14.97	4.94	0.24	1.67	21.82
18	17	Grand Theft Auto V	PS3	2013	Action	Take-Two Int	7.01	9.27	0.97	4.14	21.4
19	18	Grand Theft Auto: San Andreas	PS2	2004	Action	Take-Two Int	9.43	0.4	0.41	10.57	20.81
20	19	Super Mario World	SNES	1990	Platform	Nintendo	12.78	3.75	3.54	0.55	20.61
21	20	Brain Age: Train Your Brain in Minutes a Day	DS	2005	Misc	Nintendo	4.75	9.26	4.16	2.05	20.22
22	21	Pokemon Diamond/Pokemon Pearl	DS	2006	Role-Playing	Nintendo	6.42	4.52	6.04	1.37	18.36
23	22	Super Mario Land	GB	1989	Platform	Nintendo	10.83	2.71	4.18	0.42	18.14

Este dataset necesita de varias acciones de preprocesado antes de poder ser utilizado para el análisis, ya que contiene algunos valores nulos o erróneos que nos pueden dar lugar a problemas en los cálculos y métodos a aplicar.

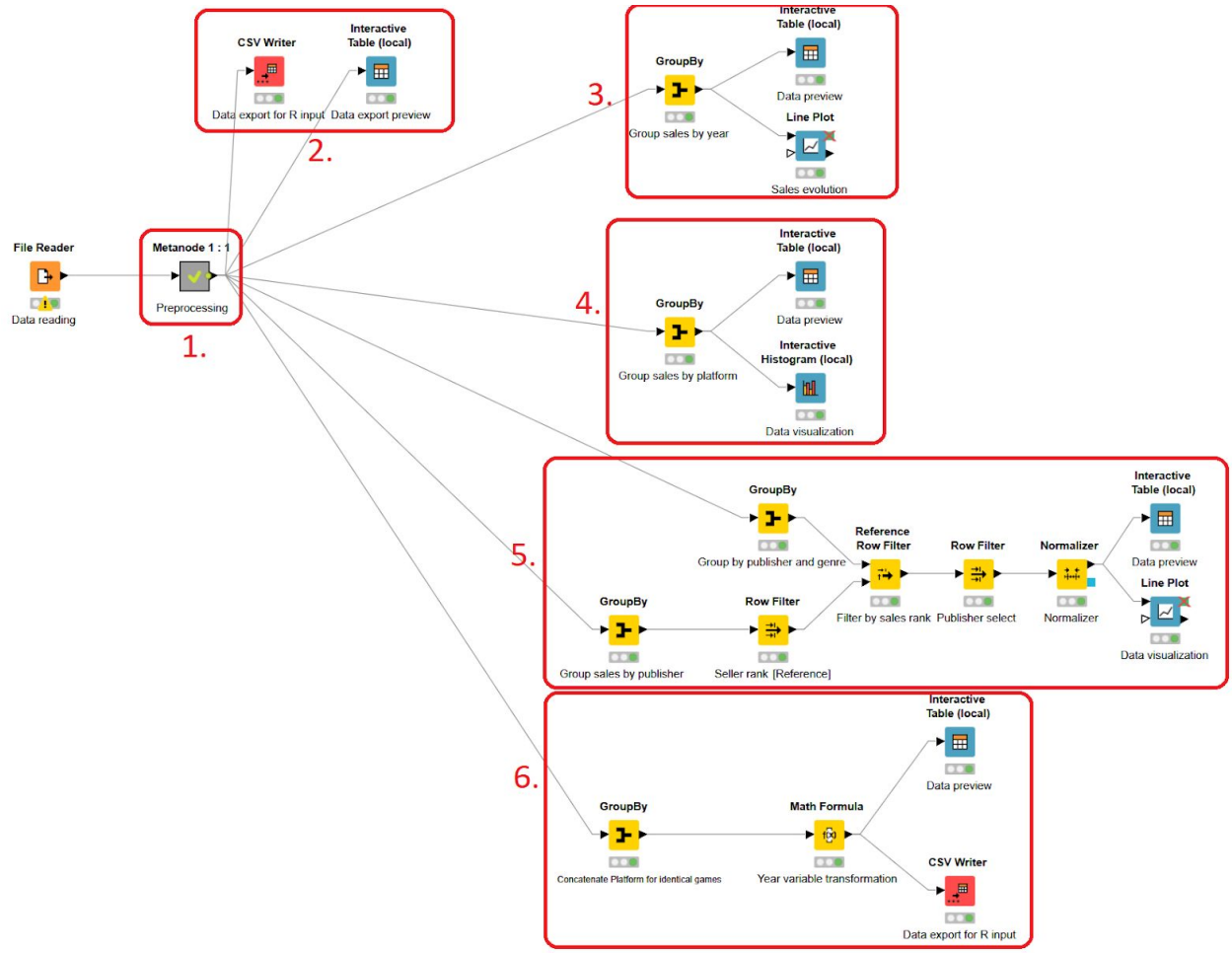
Por su parte, para el análisis no supervisado hemos usado un dataset con datos de distintos tipos de semillas de trigo, de forma que, según las características de las semillas, se pudieran diferenciar las mismas entre las tres distintas especies de trigo que recoge el conjunto.

1	A	P	C	LK	WK	A_Coef	LKG	target
2	15.26	14.84	0.871	5.763	3.312	2.221	5.22	0
3	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	0
4	14.29	14.09	0.905	5.291	3.337	2.699	4.825	0
5	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	0
6	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	0
7	14.38	14.21	0.8951	5.386	3.312	2.462	4.956	0
8	14.69	14.49	0.8799	5.563	3.259	3.586	5.219	0
9	14.11	14.1	0.8911	5.42	3.302	2.7	5	0
10	16.63	15.46	0.8747	6.053	3.465	2.04	5.877	0
11	16.44	15.25	0.888	5.884	3.505	1.969	5.533	0
12	15.26	14.85	0.8696	5.714	3.242	4.543	5.314	0
13	14.03	14.16	0.8796	5.438	3.201	1.717	5.001	0
14	13.89	14.02	0.888	5.439	3.199	3.986	4.738	0
15	13.78	14.06	0.8759	5.479	3.156	3.136	4.872	0
16	13.74	14.05	0.8744	5.482	3.114	2.932	4.825	0
17	14.59	14.28	0.8993	5.351	3.333	4.185	4.781	0
18	13.99	13.83	0.9183	5.119	3.383	5.234	4.781	0

Dado que este set de datos ya estaba preparado para el método utilizado (clusterización) no ha sido necesario el preprocesamiento en Knime de este para ser utilizado.

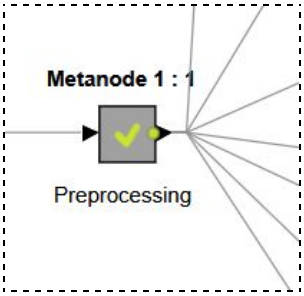
3. Preprocesamiento y Visualización

Usando la herramienta KNIME hemos realizado un preprocesamiento al dataset de ventas de videojuegos (VGSALES) que hemos utilizado para los métodos de aprendizaje supervisado. Este proceso sigue los pasos ilustrados en la imagen siguiente, tal y como pasamos a explicar y justificar brevemente a continuación.

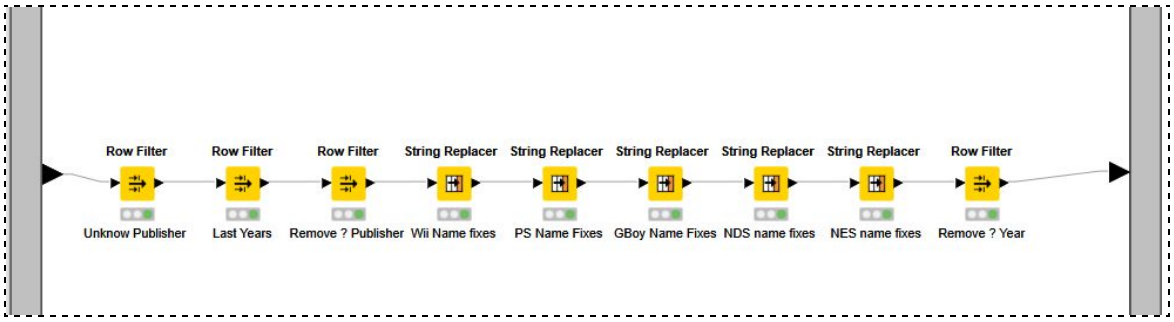


El procesamiento y visualización de Knime podemos separarlo en 6 secciones, de la forma en que están numeradas en la imagen superior.

Bloque 1:



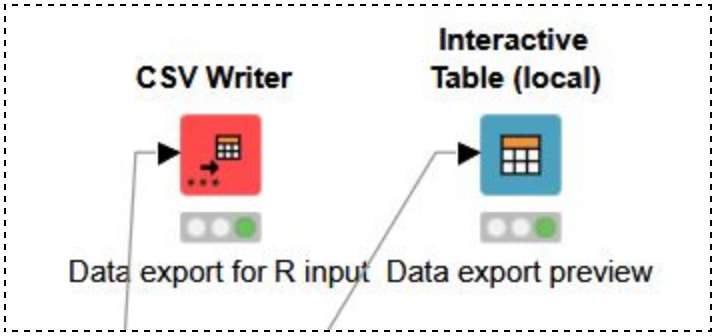
En primer lugar, tras la lectura del fichero, se realiza el preprocesamiento del archivo para **limpiar los datos erróneos** y que pueden dar lugar a errores en los cálculos. Para ello, y para dejar una vista más clara, hemos utilizado un meta-nodo, que contiene todas estas acciones de preprocesado. Si accedemos a él, podemos visualizar lo siguiente:



En este método entran los datos sin limpiar y salen ya purgados. Las acciones que se realizan dentro, por el mismo orden que aparecen en la imagen, son:

- Limpieza de datos con Publisher con valor Unknown
- Limpieza de datos de los últimos 2 años, ya que el dataset contiene muy pocos datos de estos dos períodos al no haberse recogido suficiente de estos años y puede dar lugar a confusiones a la hora de la evaluación de evolución de ventas.
- Limpieza de datos con valor "?" en la columna Publisher
- Los 5 siguientes son reemplazos de texto para que nombres de consola con diferentes valores que hacen referencia a la misma se queden con el mismo nombre.
- Finalmente, el último bloque sirve para eliminar datos donde el valor del año es "?"

Bloque 2:

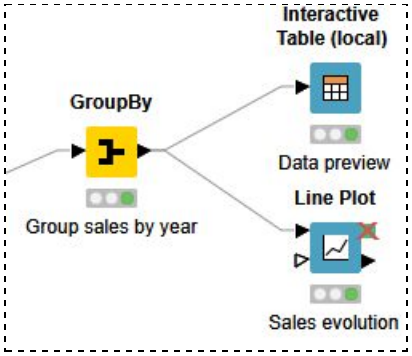


Tras la limpieza de datos, hacemos una visualización de los datos obtenidos y realizamos un export a un archivo CSV que leeremos automáticamente desde R. Este archivo va a la ruta relativa "./Datasets" de la carpeta de trabajo, que será la que utilice R para leer este archivo.

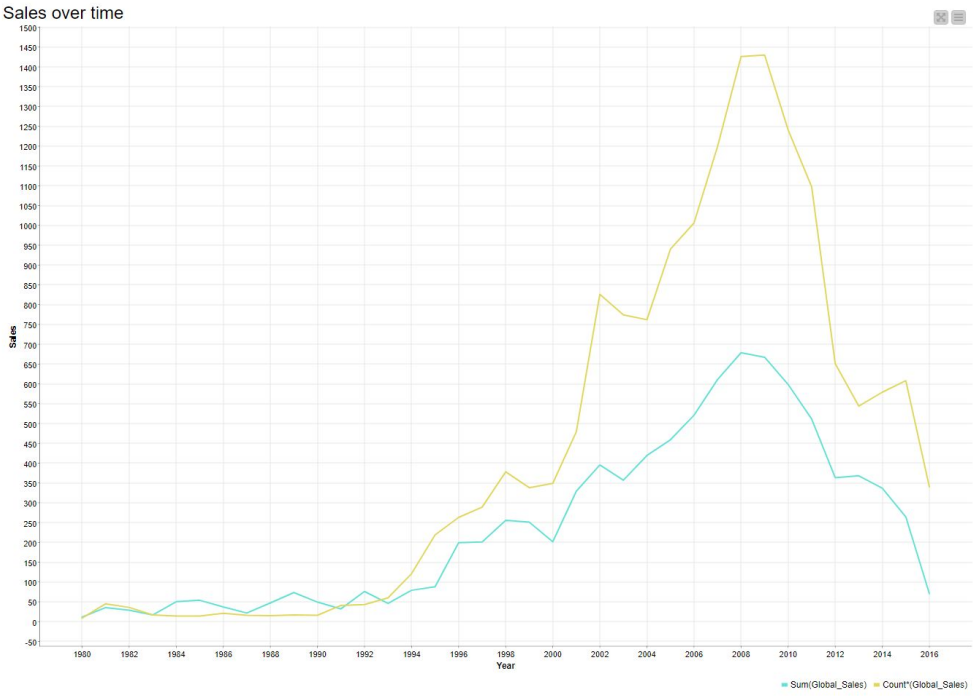
De esta forma podemos realizar cambios en Knime directamente a la vez que trabajamos con R y sin tener que modificar nada más que la ejecución en Knime.

Row ID	S Name	S Platform	I Year	S Genre	S Publisher	D NA_Sales	D EU_Sales	D JP_Sales	D Other_...	D Global_...
1	Wii Sports	Wii1	2006	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
2	Super Mario ...	Famicom	1985	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii1	2008	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports R...	Wii1	2009	Sports	Nintendo	15.75	11.01	3.28	2.96	33
5	Pokemon Re...	GBoy	1996	Role-Playing	Nintendo	11.27	8.89	10.22	1	31.37
6	Tetris	GBoy	1989	Puzzle	Nintendo	23.2	2.26	4.22	0.58	30.26
7	New Super ...	NDS	2006	Platform	Nintendo	11.38	9.23	6.5	2.9	30.01
8	Wii Play	Wii1	2006	Misc	Nintendo	14.03	9.2	2.93	2.85	29.02

Bloque 3:

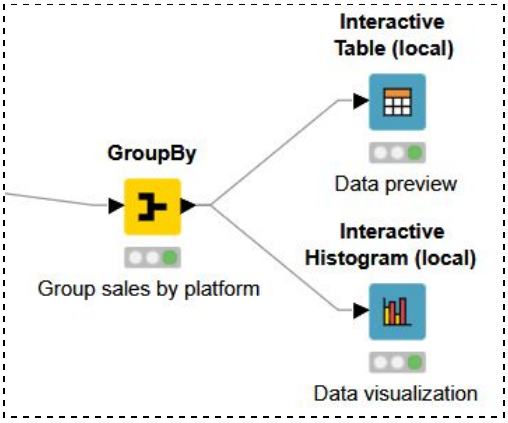


En este bloque agrupamos las ventas por año y utilizamos estos datos para visualizar cómo evolucionan las ventas de los videojuegos a lo largo de los años.



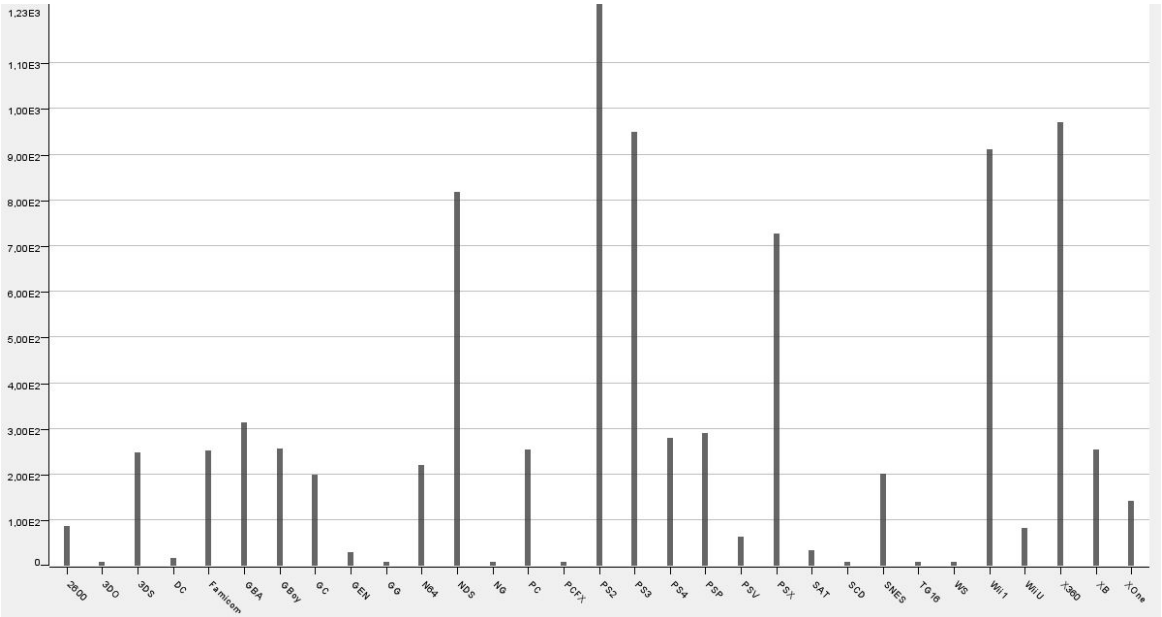
Como podemos ver, la evolución en la venta de los videojuegos (color amarillo, correpondiente a la suma de las ventas de juegos en un año) es evidente. Atribuimos la bajada final a una menor cantidad en la recolección de datos, ya que el mercado de los videojuegos continúa en alza, y sin embargo apreciamos una menor cantidad de datos (línea azul, correspondiente al número de juegos que han salido en ese año)

Bloque 4:

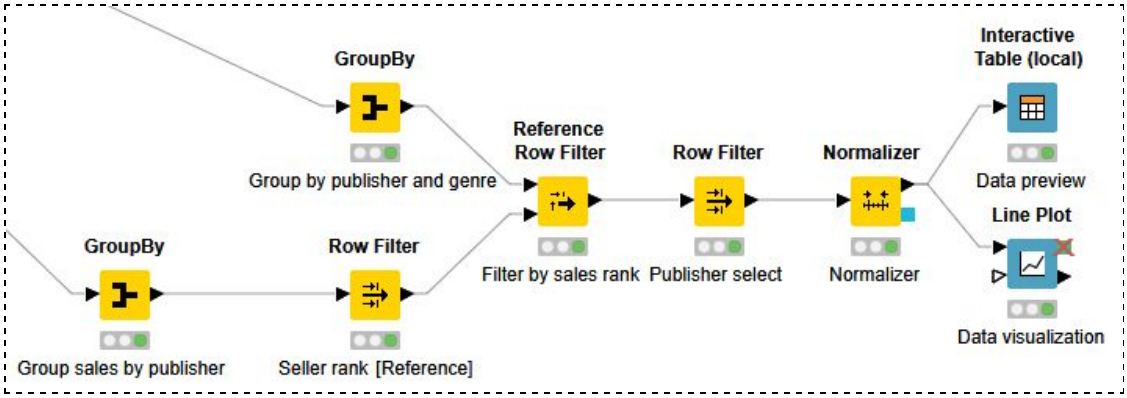


También hemos realizado una visualización de datos agrupando los datos por la plataforma en la que se vende. De esta forma podemos ver cuáles son las consolas que más éxito han tenido. Este es el caso de la PS2 y la XBOX360, que son las dos plataformas con un mayor número de ventas, como se recoge en el gráfico inferior.

Estos datos cuadran con el gráfico anteriormente visualizado de las ventas, ya que estas consolas son de la época en la que se ha producido el incremento exponencial en las ventas (años comprendidos entre 2002 y 2012.

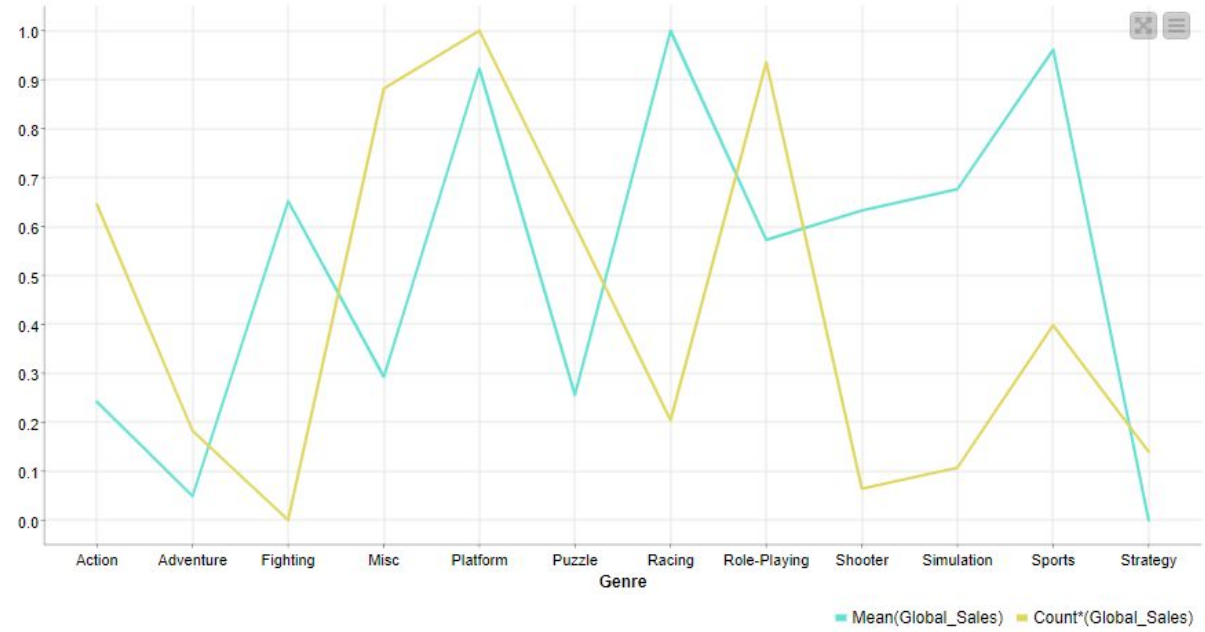


Bloque 5:



En el bloque 5, realizamos un filtrado de los mayores vendedores de juegos, agrupando los juegos por publisher y quedándonos con los 10 con mayores ventas. Posteriormente seleccionamos uno de ellos (usamos Nintendo como ejemplo en el gráfico siguiente) donde analizaremos las ventas y el número de juegos publicados, separados por las distintas categorías en las que clasificar un videojuego.

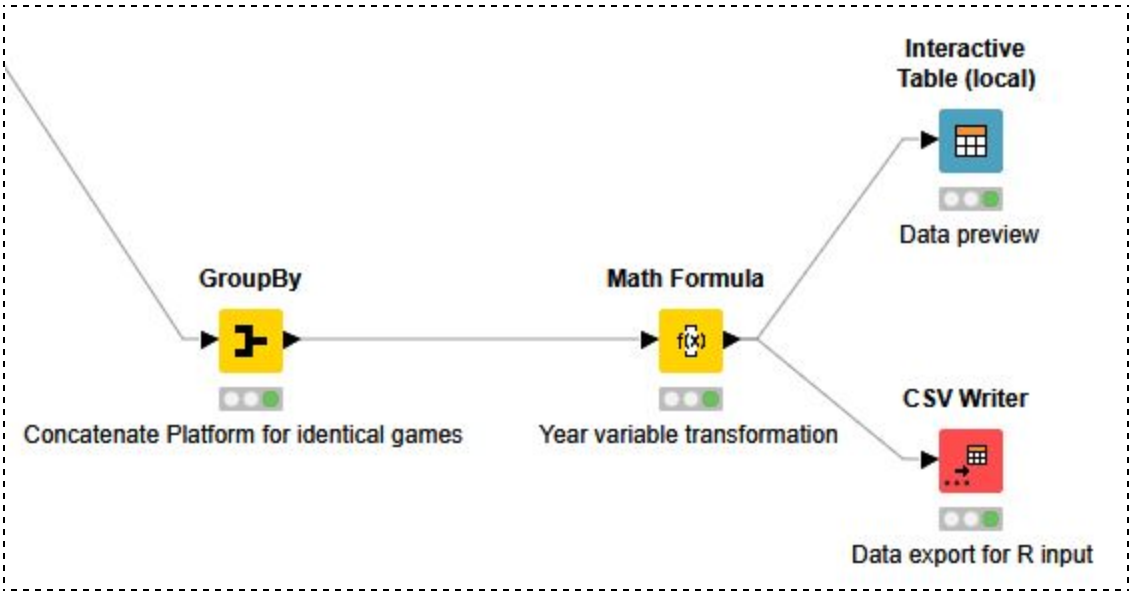
Además realizamos una normalización de los datos de número de ventas y número de publicaciones, de cara a poder realizar una comparativa más precisa, en la que ambas variables fluctúen entre 0 y 1.



Gracias a este gráfico podemos determinar las categorías donde más vende Nintendo, que es el caso de Plataformas, Carreras y Estrategia. Además, observamos que las ventas de juegos de Carreras son muy altas a pesar de que no han sacado muchos juegos de esta categoría, por lo que podríamos decir (a falta de datos que desconocemos sobre la empresa), que le interesa publicar más juegos de carreras.

Además vemos, que a pesar de sacar muchos juegos de lucha, no tienen apenas ventas, por lo que no es un mercado que a priori pueda explotar mucho.

Bloque 6:



Finalmente, con el objetivo de sacar nuevas variables que sean de utilidad a la hora de realizar el análisis supervisado, realizamos dos tratamientos de datos:

En primer lugar agrupamos los juegos con nombres idénticos pero que se publican en diferentes plataformas, y concatenamos las plataformas dentro de la columna correspondiente.

Posteriormente hacemos una conversión del año a una nueva columna, restando a 2021 el año de la publicación del juego, donde el valor es el número de años que ha pasado desde su publicación, de esta forma, también podremos tener en cuenta el tiempo que el juego lleva publicado a la hora de realizar cálculos y conclusiones.

S Name	S Concat...	I Min*(Y...	S First(G...	S First(Publis...	D Sum(N...	D Sum(EU...	D Sum(JP...	D Sum(Ot...	D Sum(Gl...	D YearCo...
'98 Koshien	PSX	1998	Sports	Magical Company	0.15	0.1	0.12	0.03	0.41	23
.hack//G.U. V...	PS2	2006	Role-Playing	Namco Bandai ...	0	0	0.17	0	0.17	15
.hack//G.U. V...	PS2	2006	Role-Playing	Namco Bandai ...	0.11	0.09	0	0.03	0.23	15
.hack//G.U. V...	PS2	2006	Role-Playing	Namco Bandai ...	0	0	0.16	0	0.16	15

4. Análisis Supervisado

Preparación del dataset para clasificación

Tras haber realizado el preprocesado en Knime como hemos descrito en la sección previa, únicamente necesitamos modificar el dataset para añadir la variable "Superventas" (Añadido como "Supersale" al estar el código en inglés).

Este parámetro lo crearemos seleccionando un porcentaje en el que queramos dividir el dataset (Porcentaje de juegos del dataset que serían superventas) y, tras definir el valor de ventas mínimas para ser un superventas, se añade a cada fila el valor booleano correspondiente (1 si es un superventas y 0 si no lo es).

```
######## Generate supersale threshold ########
# Get index
supersale_threshold_percent <- 0.5
supersale_threshold_index <- round(supersale_threshold_percent * length(vgsales_preprocessed_concatenated_platform_dummies[,1]))

# Sort by global sales
vgsales_preprocessed_concatenated_platform_dummies <-
  vgsales_preprocessed_concatenated_platform_dummies[order(-vgsales_preprocessed_concatenated_platform_dummies$Sum.Global_Sales.),]

# Get supersales threshold value
supersale_threshold <- vgsales_preprocessed_concatenated_platform_dummies[supersale_threshold_index,]$Sum.Global_Sales.

vgsales_preprocessed_concatenated_platform_dummies <- vgsales_preprocessed_concatenated_platform_dummies %>%
  mutate(Supersale = (vgsales_preprocessed_concatenated_platform_dummies$Sum.Global_Sales. >= supersale_threshold))
```

Entrenamiento y predicción

RPart

Para el primer análisis utilizaremos RPart. Aquí detallaremos el proceso que hemos seguido, mostrando así las sucesivas mejoras del modelo que hemos ido realizando.

A continuación mostramos el código utilizado para separar el dataset en entrenamiento y prueba, entrenar el árbol, predecir y por último obtener las métricas.

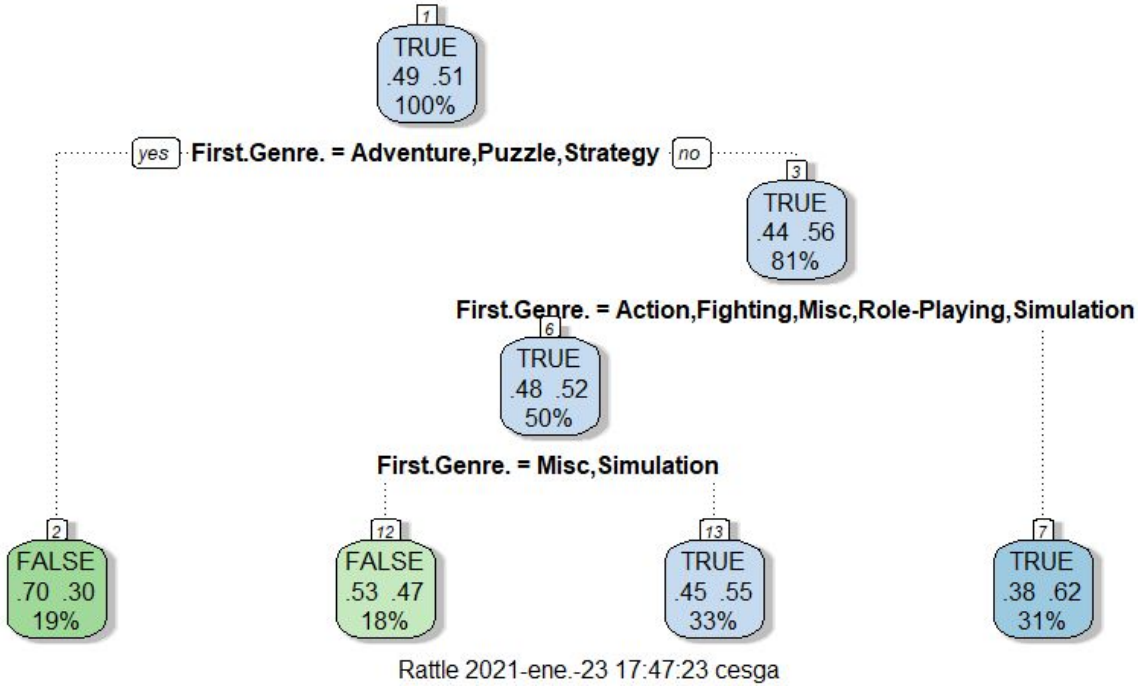
```
################ rpart Tree ################
# Split data into training and testing set
set.seed(1)
train_set_percentage <- 0.75
dt <- sort(sample(nrow(vgsales_preprocessed_concatenated_platform_dummies),
                  nrow(vgsales_preprocessed_concatenated_platform_dummies) * train_set_percentage))
train_set<-vgsales_preprocessed_concatenated_platform_dummies[dt,]
test_set<-vgsales_preprocessed_concatenated_platform_dummies[-dt,]

# Remove not useful columns
#train_set <- select(train_set, c(First.Genre., Supersale))

# Train and plot tree
tree <- rpart(Supersale ~ ., train_set, method = "class")
fancyRpartPlot(tree)

######## Metrics ########
# Predict with the test_set using the tree
pred <- predict(tree, test_set, type = "class")

# Conf. matrix creation and metrics
conf <- table(test_set$Supersale, pred)
confusionMatrix(conf)

# Summary
summary(tree)
```

El porcentaje utilizado para el set de entrenamiento es el 75% de los datos. Tras esto se seleccionan las columnas a utilizar en el entrenamiento, se crea el árbol utilizando los datos de entrenamiento y se realiza la predicción con los datos de prueba. Por último utilizamos el método “confusionMatrix” del paquete *caret* para visualizar tanto la precisión del modelo como las demás métricas distintas que ofrece, así como el método “summary”, que nos dará información del peso de las distintas variables en la clasificación final.

En las siguientes páginas veremos los distintos atributos que hemos ido utilizando para el entrenamiento y las mejoras que han ido ofreciendo cada uno de ellos, atendiendo al rendimiento que producen respectivamente.

Género

En primer lugar se ha utilizado el género para la clasificación y formación del árbol.



Como bien se puede ver, los juegos de aventura, puzzle y estrategia son directamente clasificados como no superventas (19% del dataset de entrenamiento), teniendo un error del 30%. En caso de no ser de estos 3 géneros y tampoco ser de acción, lucha, miscelánea, rol ni de simulación entraría en una hoja clasificando como superventas con un error del 38%. El restante, en caso de no pertenecer su género a miscelánea o simulación se clasificaría como superventas y en caso contrario no.

6% peso de la variable género tiene un 100% de importancia en este árbol, dando una precisión de 57,05674%.

Plataforma

Tras realizar esta clasificación, decidimos añadir la plataforma (dispositivo o consola en la que se ha publicado el videojuego) en la clasificación para intentar mejorar el modelo.

Tras haber preprocesado los datos con Knime, habiendo agrupado juegos de distintas plataformas en una fila única, concatenando dentro del valor de plataforma sus distintas apariciones (Por ejemplo, dos filas del mismo juego con plataformas “PS3” y “X360”, se unirían en una sola, con “PS3 X360” como plataforma y un valor de las ventas correspondiente la suma de sus columnas), se pasó a crear variables *dummies* para contener valores lógicos indicando las plataformas en las que el juego fue vendido.

```
######################## Extra processing ########################
# Distinct platforms
disctinct_platforms <- distinct(vgsales_preprocessed,Platform)
```

```
######## Create dummy variables for platform ########
#   if the attribute Concatenate.Platform contains the platform name previously distincted
for (i in 1:length(disctinct_platforms[,])) {
  platform_dummy_name <- paste("Platform-", disctinct_platforms[i,], sep="")

  vgsales_preprocessed_concatenated_platform_dummies <-
    vgsales_preprocessed_concatenated_platform_dummies %>%
    mutate(!!toString(platform_dummy_name) :=
             (grepl(disctinct_platforms[i,],
                    vgsales_preprocessed_concatenated_platform_dummies$Concatenate.Platform, fixed = TRUE)))
}
```

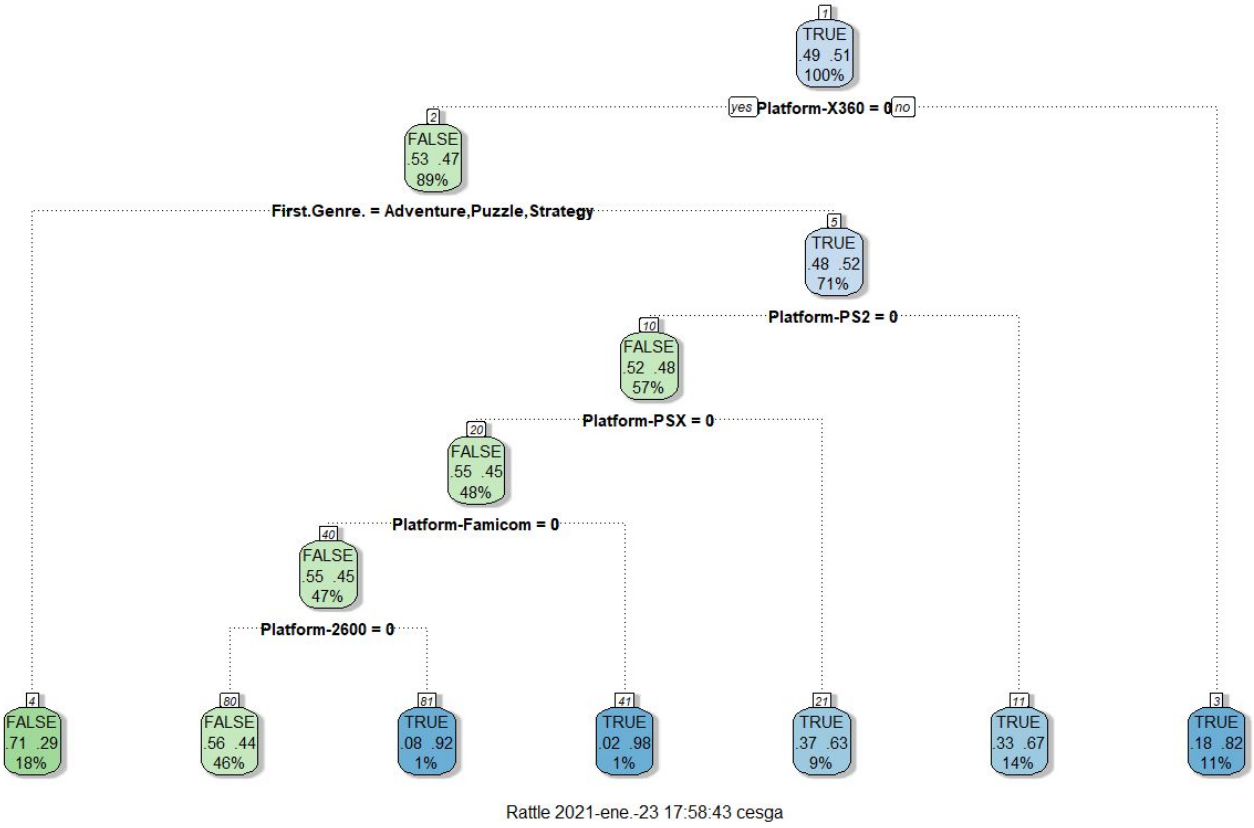
	Name	Concatenate.Platform.	Min..Year.	First.Genre.	First.Publisher.	Sum.NA_Sales.	Sum.EU_Sales.	Sum.JP_Sales.	Sum.Other_Sales.
1	Wii Sports	Wii1	2006	Sports	Nintendo	41.49	29.02	3.77	8.46
2	Grand Theft Auto V	PS3, X360, PS4, XOne, PC	2013	Action	Take-Two Interactive	23.46	23.04	1.39	8.03
3	Super Mario Bros.	Famicom, GBoy	1985	Platform	Nintendo	32.48	4.88	6.96	0.99
4	Tetris	GBoy, Famicom	1988	Puzzle	Nintendo	26.17	2.95	6.03	0.69
5	Mario Kart Wii	Wii1	2008	Racing	Nintendo	15.85	12.88	3.79	3.31
6	Wii Sports Resort	Wii1	2009	Sports	Nintendo	15.75	11.01	3.28	2.96

	Sum.Global_Sales.	YearCount	Supersale	Platform-Wii1	Platform-Famicom	Platform-GBoy	Platform-NDS	Platform-X360	Platform-PS3	Platform-PS2	Platform-SNES	Platform-GBA
1	82.74	15	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	55.92	8	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
3	45.31	36	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	35.84	33	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	35.82	13	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
6	33.00	12	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

	Platform-3DS	Platform-PS4	Platform-N64	Platform-PSX	Platform-XB	Platform-PC	Platform-2600	Platform-PSP	Platform-XOne	Platform-GC	Platform-WiiU	Platform-GEN
1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
3	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
6	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

	Platform-DC	Platform-PSV	Platform-SAT	Platform-SCD	Platform-WS	Platform-NG	Platform-TG16	Platform-3DO	Platform-GG	Platform-PCFX
1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
3	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
5	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
6	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Este fue el árbol obtenido añadiendo las nuevas variables al entrenamiento:



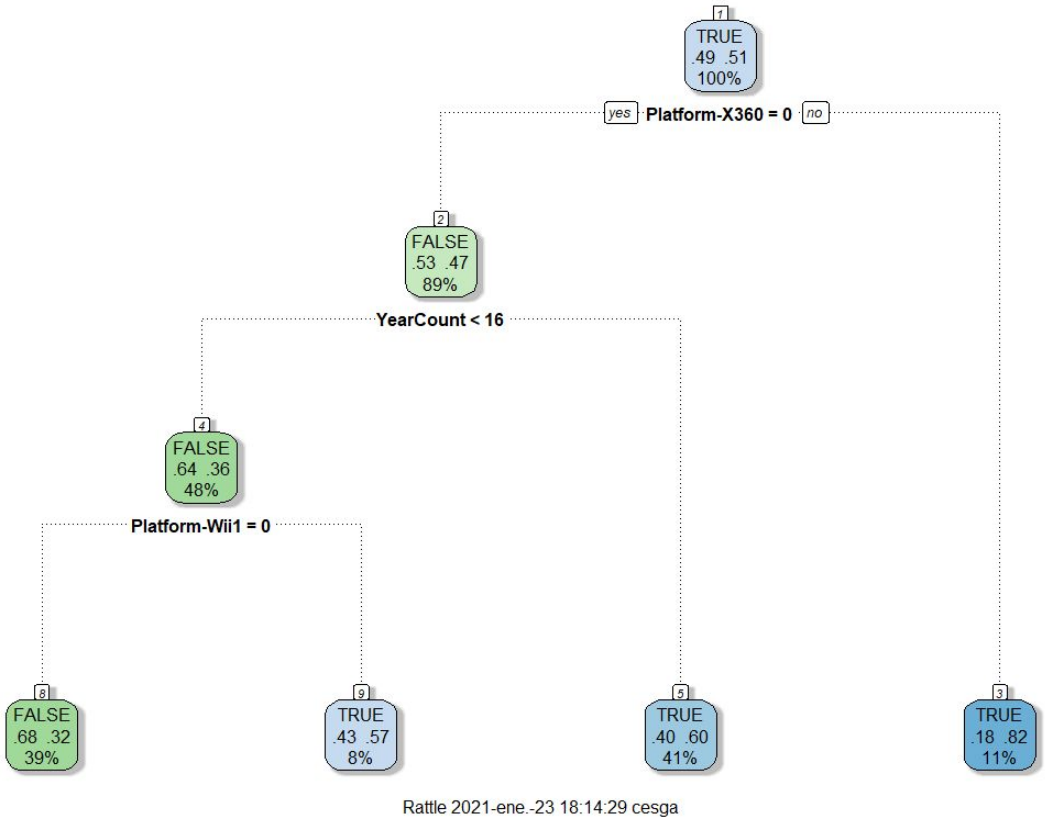
Como bien se puede ver, el género quedó reducido a un único nodo y varias plataformas entraron en juego con respecto al modelo obtenido en el paso anterior. Esta es la importancia de las distintas variables dentro del árbol:

```
Variable importance
   Platform-X360 Platform-Famicom
              35                6
    First.Genre.     Platform-2600
              22                5
    Platform-PS2       Platform-XB
              12                2
    Platform-PS3       Platform-GC
               9                1
    Platform-PSX
               7
```

Tras la predicción con los datos de prueba, la precisión del árbol fue del 63,7234%.

Diferencia de años

En este punto, se nos ocurrió crear una variable mostrando la diferencia del año actual (2021) y el año de salida del juego. Se añadió al dataset utilizando Knime con el nombre de YearCount.



Esta variable mejoró un poco el árbol, dejando atrás muchos de los nodos anteriores haciendo desaparecer completamente el género, clasificando como superventas juegos no pertenecientes a la plataforma Xbox 360 y que hayan sido publicados hace 16 años o más. Esta fue la importancia de las variables:

```
Variable importance
    YearCount Platform-X360 Platform-Wii1  Platform-PSX  First.Genre.
           28            26             9             7             7
 Platform-PS3  Platform-NDS  Platform-PS2  Platform-GBA
           7             6             4             4
```

A partir de este árbol se obtuvo una precisión en la predicción del 66,09929%.

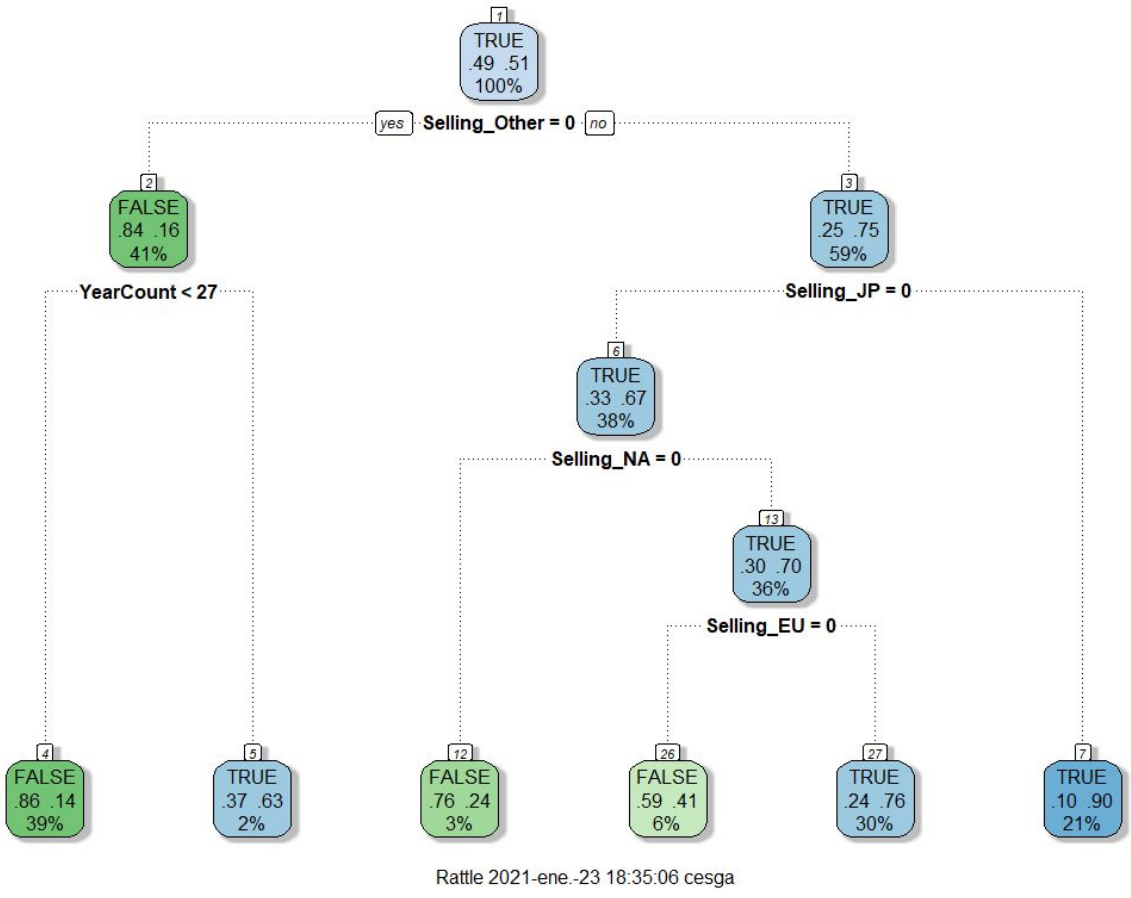
Mercado

Tras esto, quisimos ver la importancia de las ventas en los distintos mercados en la clasificación de un videojuego como superventas.

A partir de los datos de las ventas, definimos unas nuevas variables lógicas de si un juego se vende en cada mercado (Ventas en ese mercado superiores a 0.01).

```
######## Add market dummies ########
vgsales_preprocessed_concatenated_platform_dummies <-
  vgsales_preprocessed_concatenated_platform_dummies %>%
  mutate(Selling_EU = vgsales_preprocessed_concatenated_platform_dummies$Sum.EU_Sales. >= 0.01) %>%
  mutate(Selling_NA = vgsales_preprocessed_concatenated_platform_dummies$Sum.NA_Sales. >= 0.01) %>%
  mutate(Selling_JP = vgsales_preprocessed_concatenated_platform_dummies$Sum.JP_Sales. >= 0.01) %>%
  mutate(Selling_Other = vgsales_preprocessed_concatenated_platform_dummies$Sum.Other_Sales. >= 0.01)
```

Esto generó el siguiente árbol:



Como se puede ver, estos valores sustituyeron por completo los nodos antes ocupados por las distintas plataformas. La diferencia de años pasó a tener un valor de 27 en lugar de 16 (como en el árbol anterior), clasificando juegos que no se hayan vendido en otros mercados y publicados hace 27 años o más como superventas. La importancia de las variables son las siguientes:

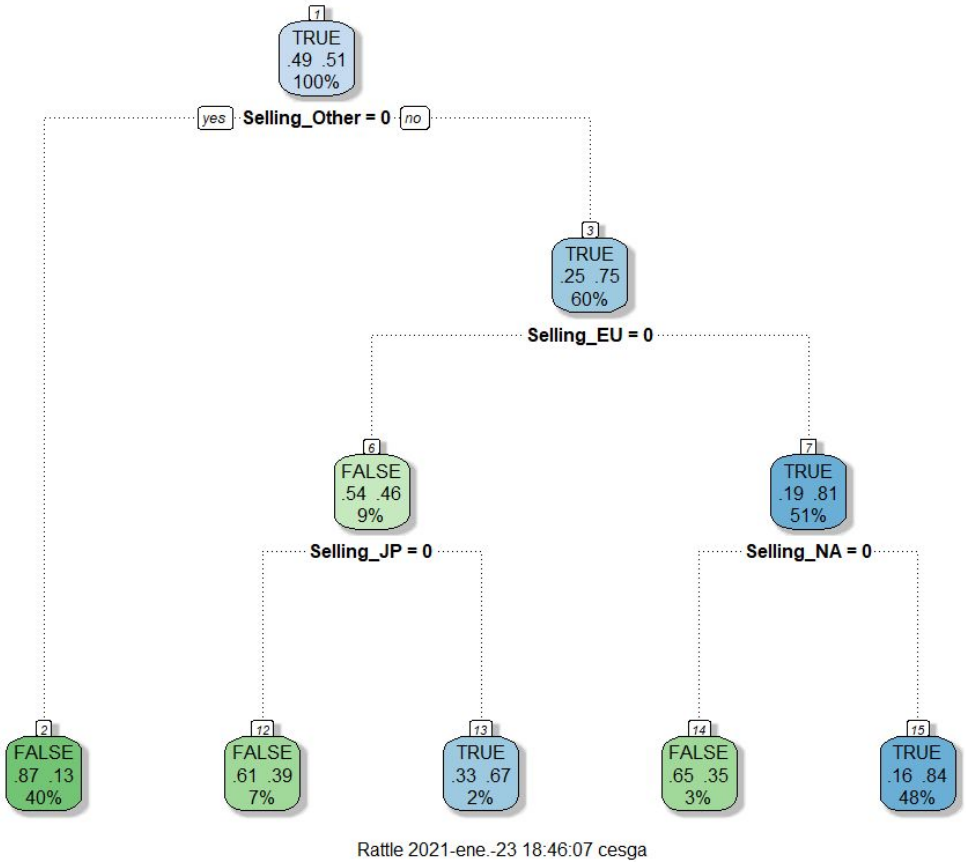
Variable importance						
Selling_Other	Selling_NA	Selling_EU	Selling_JP	First.Genre.	YearCount	Platform-SNES
38	23	23	7	3	2	2

El modelo mejoró bastante dando una precisión de 80,85106%.

Filtro de año

Por último, se decidió hacer un filtrado a los datos obteniendo las filas cuyos juegos fueron publicados entre el 2000 y el 2015 obteniendo un dataset más reciente y completo, ya que la demanda de videojuegos se ve incrementada en estos años, para así ver la repercusión en el árbol y en la clasificación de este. Además, así se evita mezclar juegos de diferentes épocas, reduciendo así los posibles sesgos que esto pudiera introducir en los modelos resultantes.

```
######## Filter by year ########
filter_year_min <- 2000
filter_year_max <- 2015
vgsales_preprocessed_concatenated_platform_dummies <-
  vgsales_preprocessed_concatenated_platform_dummies %>%
  filter(Min..Year. <= filter_year_max & Min..Year. >= filter_year_min)
```



Este paso eliminó la variable de la diferencia anual de la fórmula y modificó la posición de los nodos de mercados. Como se puede ver, el que el juego se venda en otros mercados sigue siendo el atributo más relevante. Si no se vende en otros mercados se clasifica como no superventas, acertando el 87% de las veces. El resto de nodos, el árbol clasifica como superventas juegos que se venden tanto juegos que se venden en Europa y en Norteamérica y juegos que no se venden en Europa pero sí se venden en Japón.

Esta fue la importancia de las variables en este último árbol:

```
Variable importance
Selling_Other    Selling_NA    Selling_EU
           39            25            24
   Selling_JP  First.Genre.  Platform-PSP
            5             4             2
```

Finalmente, este filtrado mejoró la precisión de la clasificación a un 82,4849%.

Estas fueron las métricas obtenidas mediante la función “confusionMatrix” de caret, comparando el primer y el último árbol de RPart:

```
Confusion Matrix and Statistics

       pred
        FALSE TRUE
  FALSE   586  776
  TRUE    435 1023

               Accuracy : 0.5706
                 95% CI : (0.5521, 0.5889)
    No Information Rate : 0.6379
    P-Value [Acc > NIR] : 1

                  Kappa : 0.133

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.5739
            Specificity : 0.5686
         Pos Pred Value : 0.4302
         Neg Pred Value : 0.7016
             Prevalence : 0.3621
         Detection Rate : 0.2078
   Detection Prevalence : 0.4830
      Balanced Accuracy : 0.5713

       'Positive' Class : FALSE
```

```
Confusion Matrix and Statistics

       pred
        FALSE TRUE
  FALSE   994  185
  TRUE    221  918

               Accuracy : 0.8248
                 95% CI : (0.8088, 0.8401)
    No Information Rate : 0.5242
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.6494

 Mcnemar's Test P-Value : 0.08238

            Sensitivity : 0.8181
            Specificity : 0.8323
         Pos Pred Value : 0.8431
         Neg Pred Value : 0.8060
             Prevalence : 0.5242
         Detection Rate : 0.4288
   Detection Prevalence : 0.5086
      Balanced Accuracy : 0.8252

       'Positive' Class : FALSE
```

En las dos imágenes anteriores podemos observar la evolución de las distintas métricas desde el comienzo al final del proceso.

Se aprecia claramente cómo la accuracy ha aumentado notablemente, desde un 57,06% que obtuvimos con el primer modelo, hasta el 82,48% final. Esto nos permite interpretar que el proceso por el que han pasado los datos ha sido eficiente, que las distintas iteraciones cobran sentido y han ido mejorando a los modelos anteriores, y que el modelo final es verdaderamente descriptivo para interpretar los datos de nuestro conjunto y obtener conclusiones útiles de ellos en base al último árbol construido.

Además de esto, es importante apuntar que la sensitividad (porcentaje de verdaderos positivos con respecto al total que el método clasifica como positivos) y la especificidad (porcentaje de verdaderos negativos sobre la suma de true negatives y false negatives) han aumentado también de forma notable, lo cual también se puede apreciar en la matriz de confusión. También se puede apreciar cómo el P-Valor se minimiza.

Por último, al aumentar el valor Kappa, nos da una cierta seguridad de que al predecir un dato, esa predicción no va a ser fruto del azar o bajo la influencia de la selección de una cierta semilla en nuestro dataset, sino que es un modelo robusto y fiable.

Naive Bayes

Para el segundo método del análisis supervisado decidimos clasificar utilizando Naive Bayes. Este es un clasificador bayesiano que utiliza internamente cálculos probabilísticos para realizar la clasificación, obteniendo el resultado en base a las probabilidades condicionadas para cada valor de la variable. Aún pareciendo su diseño ingenuo y su definición simple, estos clasificadores han funcionado bastante bien en multitud de situaciones reales complejas.

```
######## Naive Bayes Tree ########
# Split data into training and testing set
set.seed(1)
dt <- sort(sample(nrow(vgsales_preprocessed_concatenated_platform_dummies),
                  nrow(vgsales_preprocessed_concatenated_platform_dummies) * train_set_percentage))
train_set<-vgsales_preprocessed_concatenated_platform_dummies[dt,]
test_set<-vgsales_preprocessed_concatenated_platform_dummies[-dt,]

# Train and plot tree
tree <- naiveBayes(Supersale ~ ., train_set, type = "class")

######## Metrics ########
# Predict with the test_set using the tree
pred <- predict(tree, test_set, type = "class")

# Conf. matrix creation
conf <- table(test_set$Supersale, pred)

# Accuracy metric
acc <- sum(diag(conf)) / sum(conf)
print(acc)
```

Utilizando el mismo dataset modificado del último punto de la clasificación con RPart, este método nos arrojó una precisión de 80,8887%.

```
Confusion Matrix and Statistics

       pred
        FALSE TRUE
  FALSE   918  261
  TRUE    182  957

               Accuracy : 0.8089
                 95% CI : (0.7923, 0.8247)
    No Information Rate : 0.5255
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6181

 Mcnemar's Test P-Value : 0.0002106

            Sensitivity : 0.8345
            Specificity : 0.7857
         Pos Pred Value : 0.7786
         Neg Pred Value : 0.8402
             Prevalence : 0.4745
         Detection Rate : 0.3960
   Detection Prevalence : 0.5086
      Balanced Accuracy : 0.8101

       'Positive' Class : FALSE
```

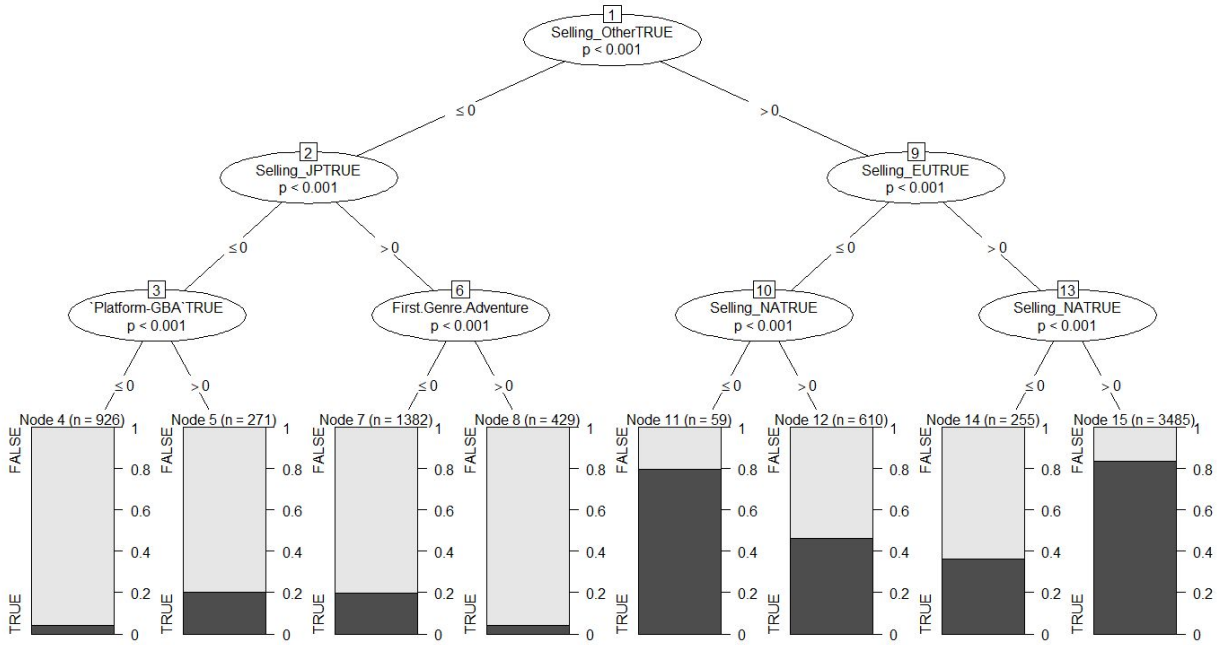
Analizando estas métricas, en contraposición con las de RPart, se puede ver que tiene una mayor sensibilidad pero una peor especificidad. Aún teniendo una precisión inferior a RPart, estos valores indican que tiene una mayor precisión para acertar los positivos pero arroja una mayor cantidad de falsos negativos, descendiendo su accuracy por este motivo.

El valor Kappa por su parte nos indica que estos resultados pueden fluctuar en mayor medida que RPart dependiendo de la semilla que se elija, al tener un valor de esta métrica inferior al obtenido anteriormente con RPart.

CTree2

CTree2 es un algoritmo recursivo de partición binaria el cual utiliza modelos de regresión usando una bien definida teoría de inferencia condicional. Se puede aplicar a todo tipo de problemas de regresión incluyendo problemas nominales, ordinales, numéricos, o incluso de variables de respuesta multivariante y escalas de medición arbitraria de los covariantes. Este fue el que utilizamos como tercer método.

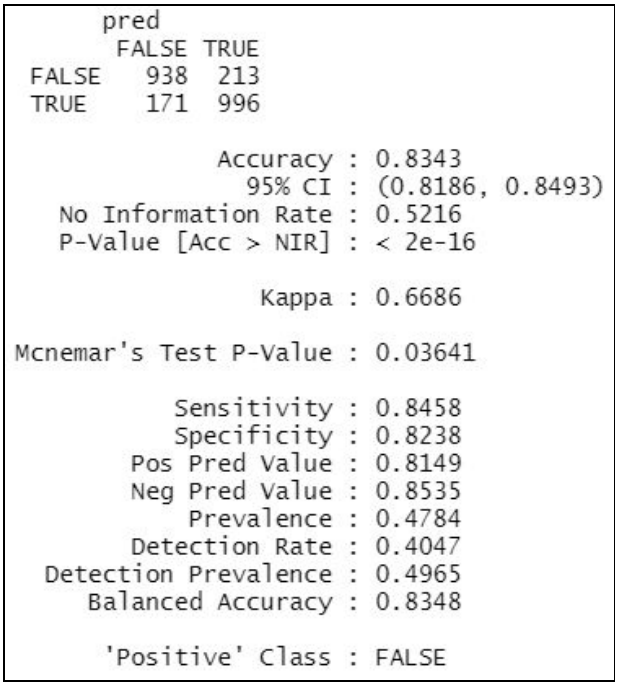
```
######## Ctree2 Tree ########
# Split data into training and testing set
set.seed(1)
dt <- sort(sample(nrow(vgsales_preprocessed_concatenated_platform_dummies),
                  nrow(vgsales_preprocessed_concatenated_platform_dummies) * train_set_percentage))
train_set<-vgsales_preprocessed_concatenated_platform_dummies[dt,]
test_set<-vgsales_preprocessed_concatenated_platform_dummies[-dt,]

train_set <- train_set %>%
  mutate(Supersale = factor(train_set$Supersale))
test_set <- test_set %>%
  mutate(Supersale = factor(test_set$Supersale))

tree <- train(
  Supersale ~., data = train_set, method = "ctree2",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(maxdepth = 3, mincriterion = 0.95 )
)

plot(model$finalModel)

######## Metrics ########
# Predict with the test_set using the tree
pred <- predict(tree, test_set, type = "raw")

# Conf. matrix creation
conf <- table(test_set$Supersale, pred)

# Accuracy metric
acc <- sum(diag(conf)) / sum(conf)
print(acc)
```



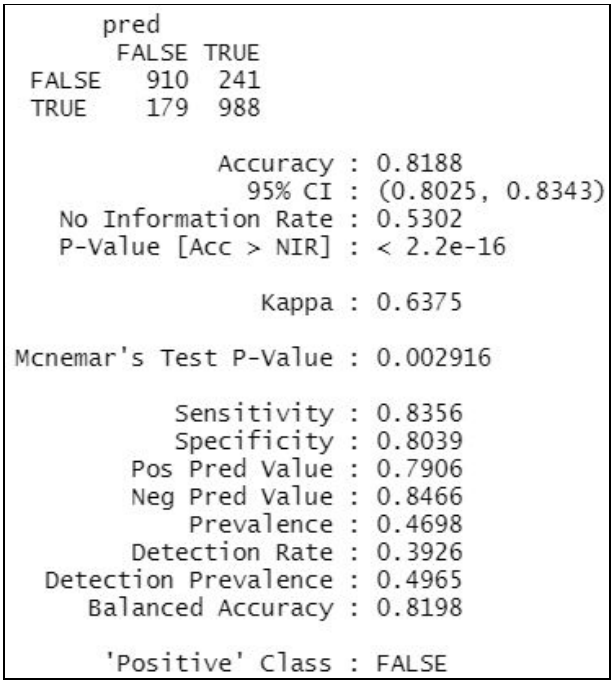
Utilizando los mismos datos que en el último punto del árbol de RPart y el usado con Naive Bayes la precisión es de 82,4849%.

```
Confusion Matrix and Statistics

       pred
        FALSE TRUE
  FALSE   994  185
  TRUE    221  918

               Accuracy : 0.8248
                 95% CI : (0.8088, 0.8401)
    No Information Rate : 0.5242
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.6494

 Mcnemar's Test P-Value : 0.08238

            Sensitivity : 0.8181
            Specificity : 0.8323
         Pos Pred Value : 0.8431
         Neg Pred Value : 0.8060
             Prevalence : 0.5242
         Detection Rate : 0.4288
   Detection Prevalence : 0.5086
      Balanced Accuracy : 0.8252

       'Positive' Class : FALSE
```

Estos valores son los mismos que los de la última parte de RPart. Esto se debe a que tanto rpart como ctree realizan splits binarios recursivamente basados en los valores del conjunto de covariantes. Aunque RPart normalmente utiliza la información de las variables exclusivamente en la selección del covariante y, por el contrario, CTree2 tiende a elegir variables con mayor posibilidad de división en un futuro, el árbol producido es el mismo dada las restricciones del problema y dataset y el peso de las variables utilizadas. Utilizando una semilla distinta obtenemos resultados distintos:



```
      pred
       FALSE TRUE
 FALSE   938  213
 TRUE    171  996

               Accuracy : 0.8343
                 95% CI : (0.8186, 0.8493)
    No Information Rate : 0.5216
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.6686

 Mcnemar's Test P-Value : 0.03641

            Sensitivity : 0.8458
            Specificity : 0.8238
         Pos Pred Value : 0.8149
         Neg Pred Value : 0.8535
             Prevalence : 0.4784
         Detection Rate : 0.4047
   Detection Prevalence : 0.4965
      Balanced Accuracy : 0.8348

       'Positive' Class : FALSE
```



```
      pred
       FALSE TRUE
 FALSE   910  241
 TRUE    179  988

               Accuracy : 0.8188
                 95% CI : (0.8025, 0.8343)
    No Information Rate : 0.5302
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6375

 Mcnemar's Test P-Value : 0.002916

            Sensitivity : 0.8356
            Specificity : 0.8039
         Pos Pred Value : 0.7906
         Neg Pred Value : 0.8466
             Prevalence : 0.4698
         Detection Rate : 0.3926
   Detection Prevalence : 0.4965
      Balanced Accuracy : 0.8198

       'Positive' Class : FALSE
```

Se puede ver cómo con esta otra semilla, la precisión de CTree2 (imagen derecha) tiene una menor precisión que RPart.

Conclusiones

En cuanto a la comparativa de los 3 métodos utilizados (RPart, Naive Bayes y CTree2), tras afinar la semilla en el último paso, RPart nos ha proporcionado la mayor precisión de los 3 métodos (83,43%) aunque los 3 han quedado bastante parejos.

Como se ha podido ver, los mercados en los que los videojuegos han sido vendidos han generado un mejor modelo, siendo las ventas en otros mercados no principales el que más importancia toma en la clasificación de si es un superventas o no. Esto puede ser porque juegos más importantes, con un mayor presupuesto y considerados superventas, llegan a estos mercados y los juegos de empresas más pequeñas no lo consiguen.

También la variable de la diferencia de años entra en juego clasificando como superventas juegos antiguos, ya que antiguamente no se publican tantos juegos como ahora (dado el abaratamiento de los costes en hardware y la aparición de motores gráficos gratuitos para la programación de videojuegos y el incremento de la demanda del sector en los últimos años), consiguiendo éstos ventas superiores al tener un catálogo más reducido del que elegir.

El filtrado de año incluido en la última parte reduce el número de instancias del dataset, pero también mejora la clasificación al tener datos más parejos entre sí y menos afectados por el cambio cultural y social en la industria del videojuego en los finales del siglo XX y principios del XXI también ocasionado por el auge de las comunicaciones y las mejoras en los dispositivos móviles así como la aparición y adopción de internet globalmente.

Por estos motivos, creemos que el último modelo obtenido con RPart es el que proporciona una mayor información y unas conclusiones más cercanas a la realidad (añadiendo al problema el conocimiento que no queda recogido en los datos, pero que nosotros como usuarios podemos aportar desde fuera, como por ejemplo esa evolución de la industria explicada anteriormente). Además de retornar un modelo que encaja con lo que nosotros esperábamos en base a nuestras ideas y conocimiento previo, es el método que devuelve un mejor rendimiento, siendo por tanto el que destacaríamos de entre los tres realizados.

5. Análisis no Supervisado

Durante la realización de este análisis se aplicaron diferentes técnicas de clustering sobre un dataset formado por distintos tipos de semillas de trigo, de forma que, según las características de las semillas, se pudieran diferenciar las mismas en distintos tipos de trigo (tipos 0, 1 y 2 según el dataset).

Como parte del preprocesado general, se renombraron las columnas

```
# rename column and correcting data type
names(seed) <- c("Area", "Perimeter", "Compactness",
                 "Length", "Width", "Asymetry.coef",
                 "Grove.length", "Type")
```

Tras esto, se le sumó 1 a todos los tipos, de forma que los tipos pasaron a ser tipos 1, 2 y 3:

```
seed$Type <- as.numeric(seed$Type+1)
```

Kmeans

Kmeans es un algoritmo de clasificación no supervisada, que agrupa objetos en K clusters basándose en sus características. Este agrupamiento tiene lugar minimizando la suma de las distancias entre cada objeto y el centroide del cluster al que pertenece. El algoritmo Kmeans requiere generalmente del uso de la distancia Euclídea para la obtención de soluciones eficientes.

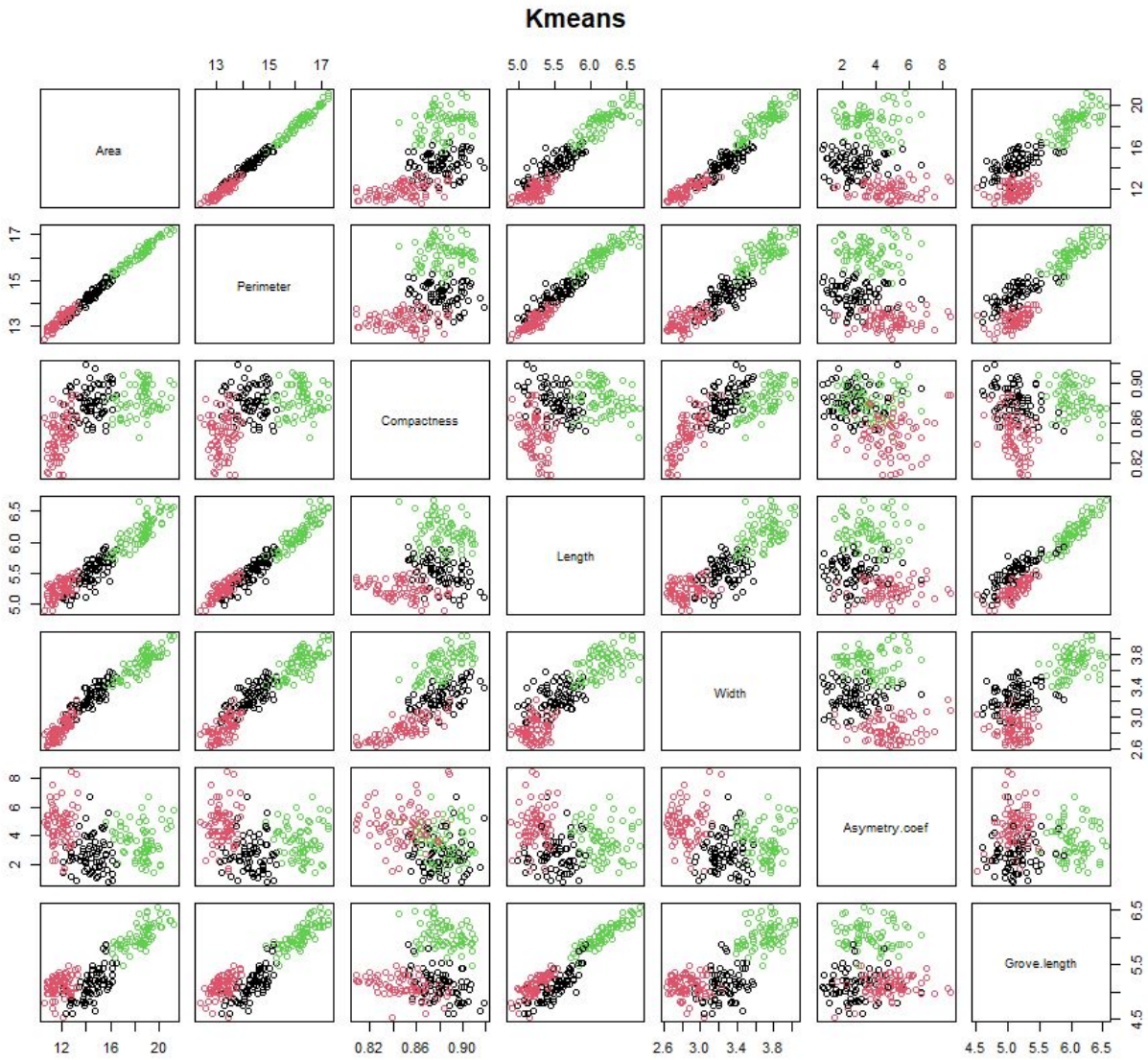
Para aplicar esta técnica se tuvieron en cuenta todas las variables excepto el tipo de semilla, para lo cual tuvimos que eliminar esa columna:

```
seed <- select(seed, -c('Type'))
```

Tras esto, aplicamos el algoritmo Kmeans al dataset, indicando que queremos obtener 3 clusters, ya que sabemos de antemano que es el número de tipos de semillas que tenemos:

```
# Calculate kmeans
km_seed<-kmeans(scale(seed),centers = 3, nstart = 20)
```

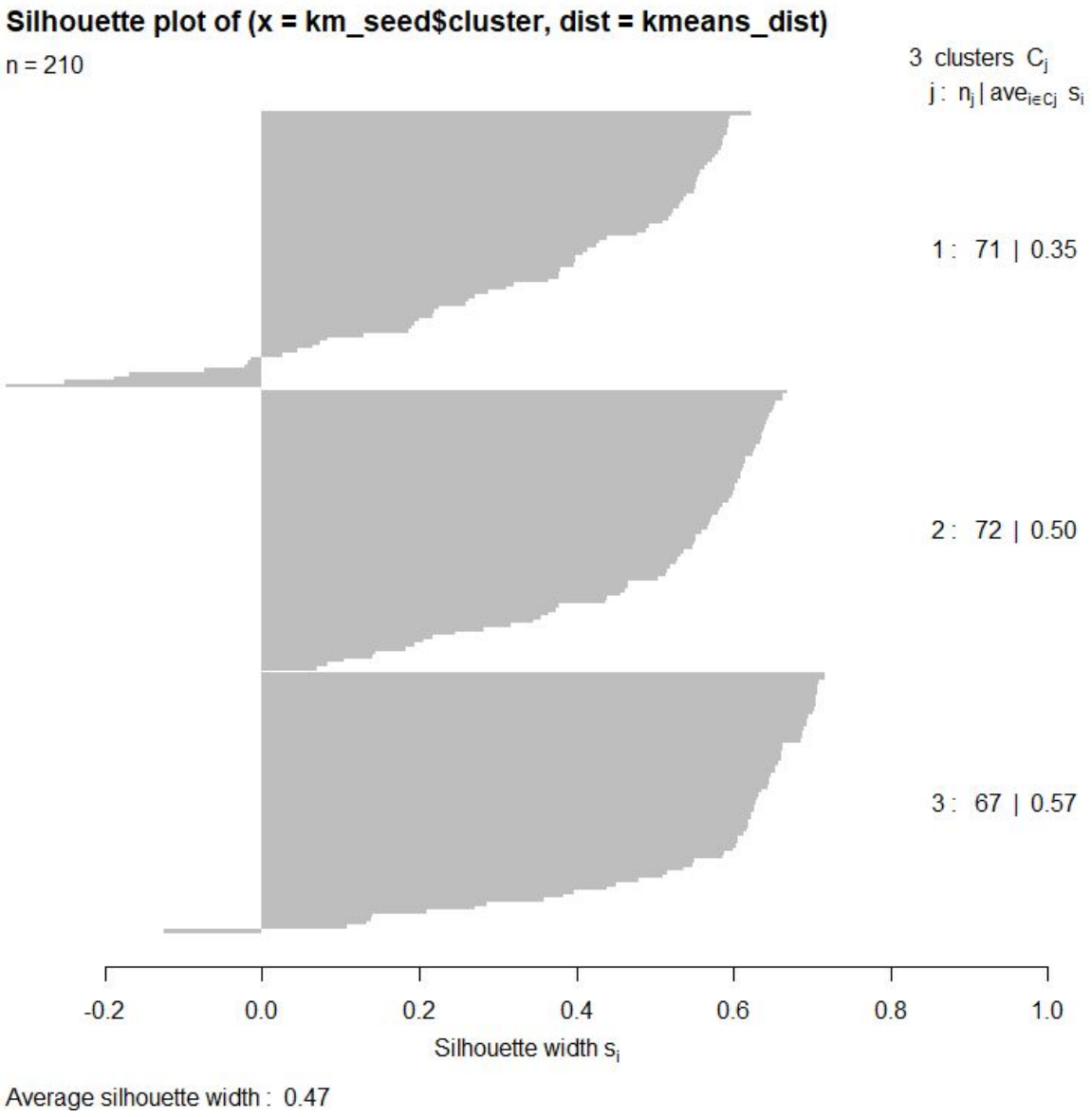
Tras hacer el plot del resultado de este algoritmo, obtenemos los siguientes resultados, en los que podemos ver 3 que los clusters están bastante diferenciados entre sí:



Después, calculamos la gráfica Silhouette para analizar la distancia entre los clusters y la compacidad de los mismos:

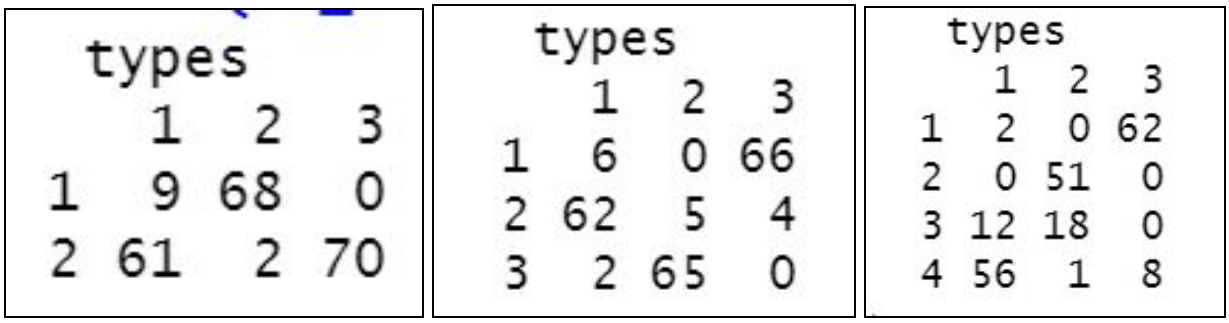
```
#Silhouette
kmeans_dist = vegdist(seed)
kmeans_sil = silhouette (km_seed$cluster,kmeans_dist)
windows()
plot(kmeans_sil)
```

Con lo que obtuvimos los siguientes resultados:



Como se puede observar en la imagen que muestra el resultado del algoritmo, si examinamos las diferentes gráficas podemos advertir que, en la gran mayoría de representaciones, cada uno de los 3 clusters en los que se ha separado tiene las fronteras bastante marcadas, por ejemplo en las gráficas Área/Perímetro, Área/Anchura, etc.

Pese a obtener buenos resultados, y saber a priori que el parámetro k tenía que tener un valor 3, ya que conocíamos de antemano que trabajamos con tres especies diferentes de trigo, hicimos la prueba con los valores k = 2 y k = 4. Con ello, obtuvimos los resultados reflejados en las matrices de confusión mostradas a continuación:



types	1	2	3
1	9	68	0
2	61	2	70

types	1	2	3
1	6	0	66
2	62	5	4
3	2	65	0

types	1	2	3
1	2	0	62
2	0	51	0
3	12	18	0
4	56	1	8

Comenzando por la izquierda, podemos ver cómo al usar k = 2, agrupa en un mismo cluster a los elementos de la especie 1 y a los de la especie 3, aislando a los de la especie 2 en un clúster sólo para este tipo. Por su parte, al usar k = 4 (matriz de la derecha), podemos apreciar cómo trata de subdividir las especies en varios subgrupos diferentes; por ejemplo, los de la especie 1 los reparte entre el cluster 3 y el 4, o los de la especie 2 los distribuye entre los cluster 3 y 4. Sin embargo, no hay diferencias significativas entre los elementos de un mismo cluster como para separarlos en dos subgrupos.

Como podemos apreciar en la matriz de confusión central (k = 3), es con diferencia la que mejor clasifica y la que menos falla. Con esto podemos afirmar que la división en 3 clusters es correcta, que el algoritmo K-medias clasifica y discrimina bien entre los tres tipos de semilla, coincidiendo con lo que esperábamos a priori, al conocer el número de especies de trigo con el que estábamos trabajando.

Kmedioids

El algoritmo Kmedioids es un algoritmo de clustering similar al Kmeans en su esencia, ya que siguen una misma estructura algorítmica. Su gran diferencia es que Kmedioids elige puntos de datos reales como centro de cada cluster (lo que llamamos medioide), a diferencia del Kmeans, que elige el punto medio del conjunto de datos, sin importar si ese punto corresponde con una instancia del dataset o no. Esto aumenta la facilidad a la hora de interpretar los centros de los clusters, ya que rápidamente podremos hacer la correspondencia con datos reales. Además, Kmedioids no está "atado" al uso de una distancia como la euclídea para la obtención de resultados eficientes, y debido a que se minimiza una suma de diferencias por pares en lugar de una suma de distancias euclídeas al cuadrado, es más robusto al ruido que el Kmeans.

Tras el Kmeans, decidimos aplicar Kmedioids usando el método pam de la librería cluster:

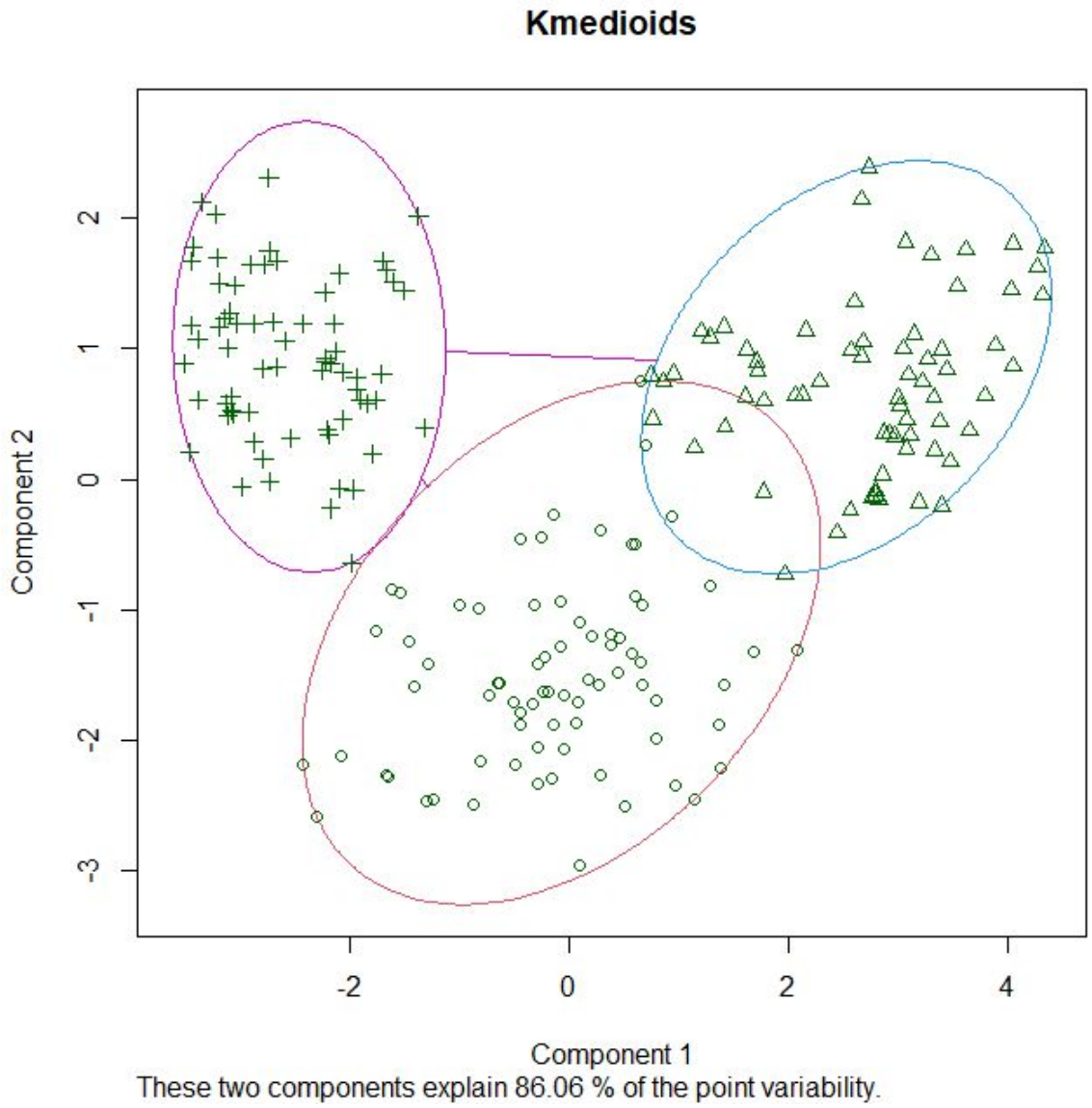
```
# Calculate pam
km_seed_kmd<-pam(scale(seed_kmd), 3)
```

De nuevo, decidimos aplicarlo con 3 clusters ya que conocíamos de antemano el número de tipos diferentes de semilla.

Después de esto, realizamos el plot para el resultado del algoritmo:

```
plot(km_seed_kmd,col=(km_seed_kmd$cluster)*10, main="Kmedioids")
```

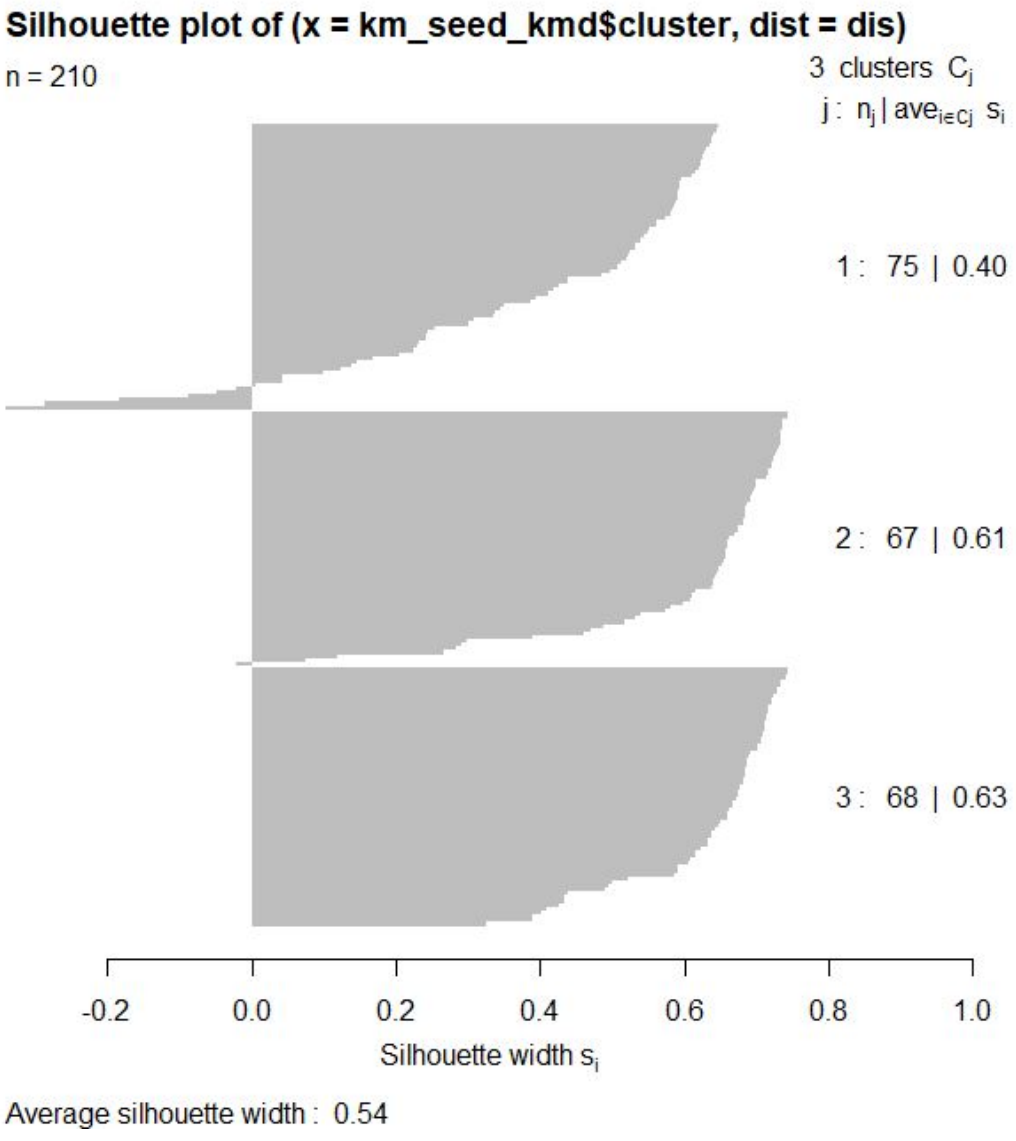
Con lo que obtuvimos el siguiente resultado:



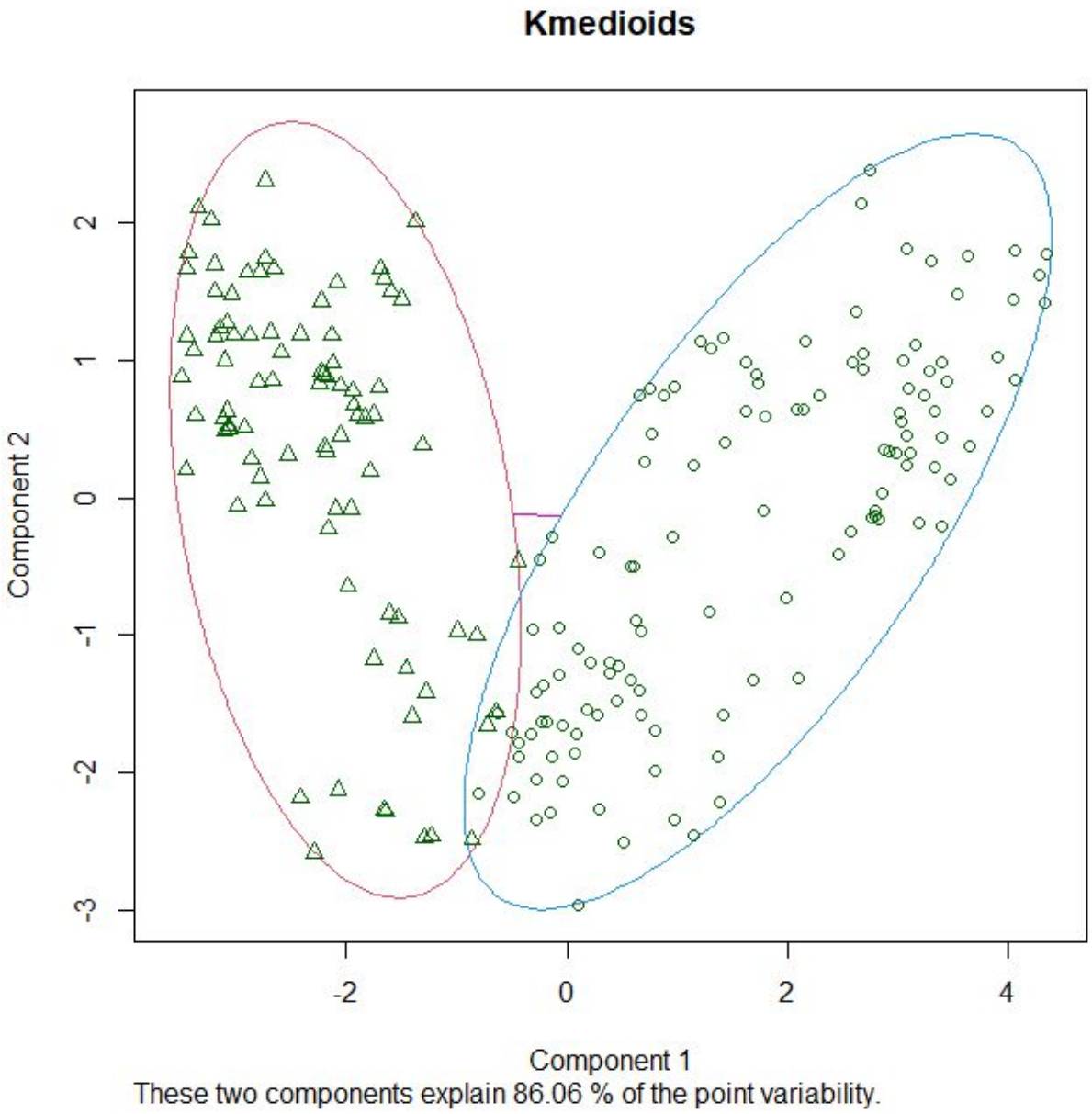
Después de conocer los resultados, mostramos la gráfica Silhouette para este caso usando el siguiente código:

```
#Silhouette
dis = vegdist(seed_kmd)
sil = silhouette (km_seed_kmd$cluster,dis)
windows()
plot(sil)
```

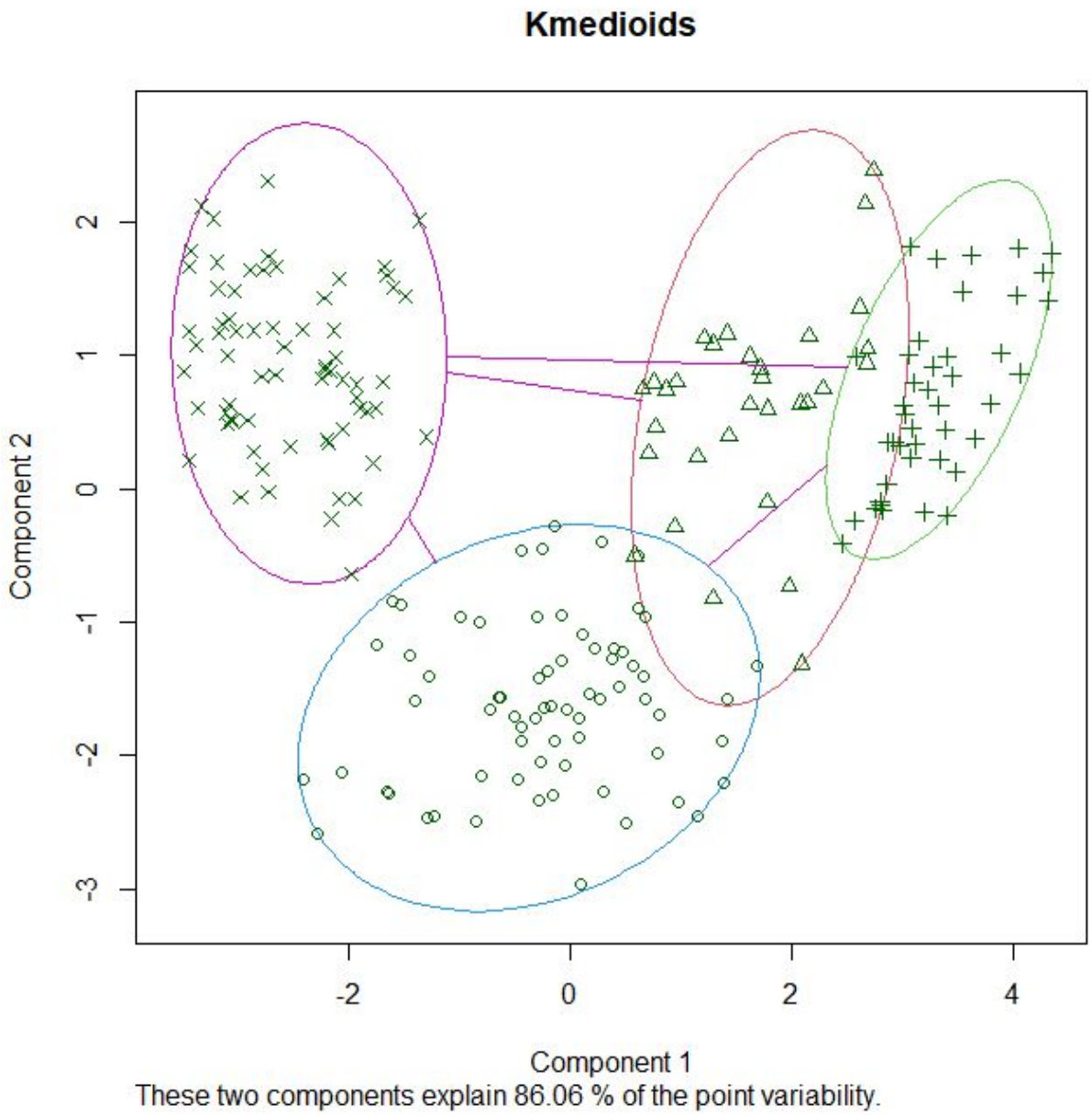
Con lo que obtuvimos estos resultados:



Como prueba extra, repetimos el Kmedioids con 2 y 4 clusters:

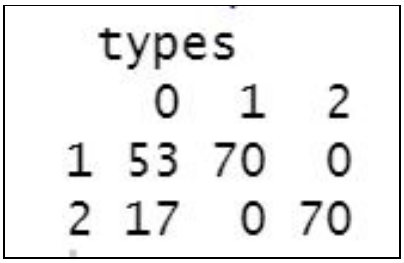


Como se puede observar, usando únicamente 2 clusters, el grupo que tenemos abajo, que se encuentra bastante separado del resto, se reparte entre cada uno de ellos, dejando huérfano el tercer tipo de semilla.

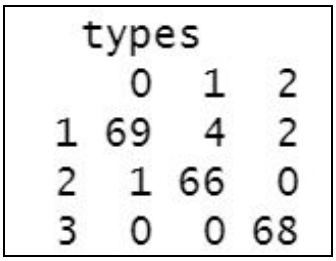


Al repetir el experimento usando 4 clusters, observamos que el grupo de abajo se establece como un único cluster, y el grupo de arriba a la derecha se ve separado en 2, pese a no haber una distancia clara entre ellos que motive a realizar la separación.

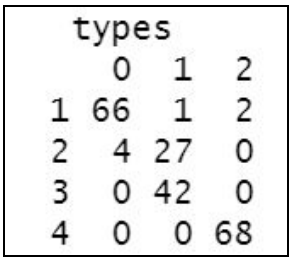
Hemos calculado las matrices de confusión para el método para los distintos valores de k (2, 3 y 4).



types			
	0	1	2
1	53	70	0
2	17	0	70



types			
	0	1	2
1	69	4	2
2	1	66	0
3	0	0	68



types			
	0	1	2
1	66	1	2
2	4	27	0
3	0	42	0
4	0	0	68

Con eso, hemos podido reafirmar que agrupar en 3 clusters es la opción correcta, ya que con k = 2, los tipos 0 y 1 se unen en un único cluster enorme, y con k = 4, el tipo 1 se encuentra dividido entre los clusters 2 y 3, de la misma forma que ocurría en el caso del K-medias.

Es por ello que podemos confirmar que el método Kmedioids separa el dataset en 3 grupos bien definidos, y que ayudan de forma exitosa en la diferenciación de los diferentes tipos de semillas que tenemos, coincidiendo la salida del método con lo esperado, gracias a conocer a priori el número de especies de trigo distintas con las que estamos trabajando.

Hierarchical clusterization

Tras aplicar las técnicas anteriores, decidimos usar por último un método de clustering jerárquico, en este caso, el algoritmo **Agglomerative Nesting**.

El Agglomerative Nesting (También conocido como **AGNES**) es el tipo más común de clustering jerárquico utilizado para agrupar objetos en función de su similitud. El algoritmo comienza tratando cada objeto como un clúster único, a continuación, se fusionan pares de clusters, comenzando por aquellos con mayor parecido. Estas agrupaciones se realizan sucesivamente hasta que todos los clusters han sido fusionados en un único cluster que los contiene a todos. El resultado de este algoritmo es una representación basada en árboles llamada **dendograma**.

Para su aplicación, utilizamos el método agnes de la librería cluster de la siguiente forma:

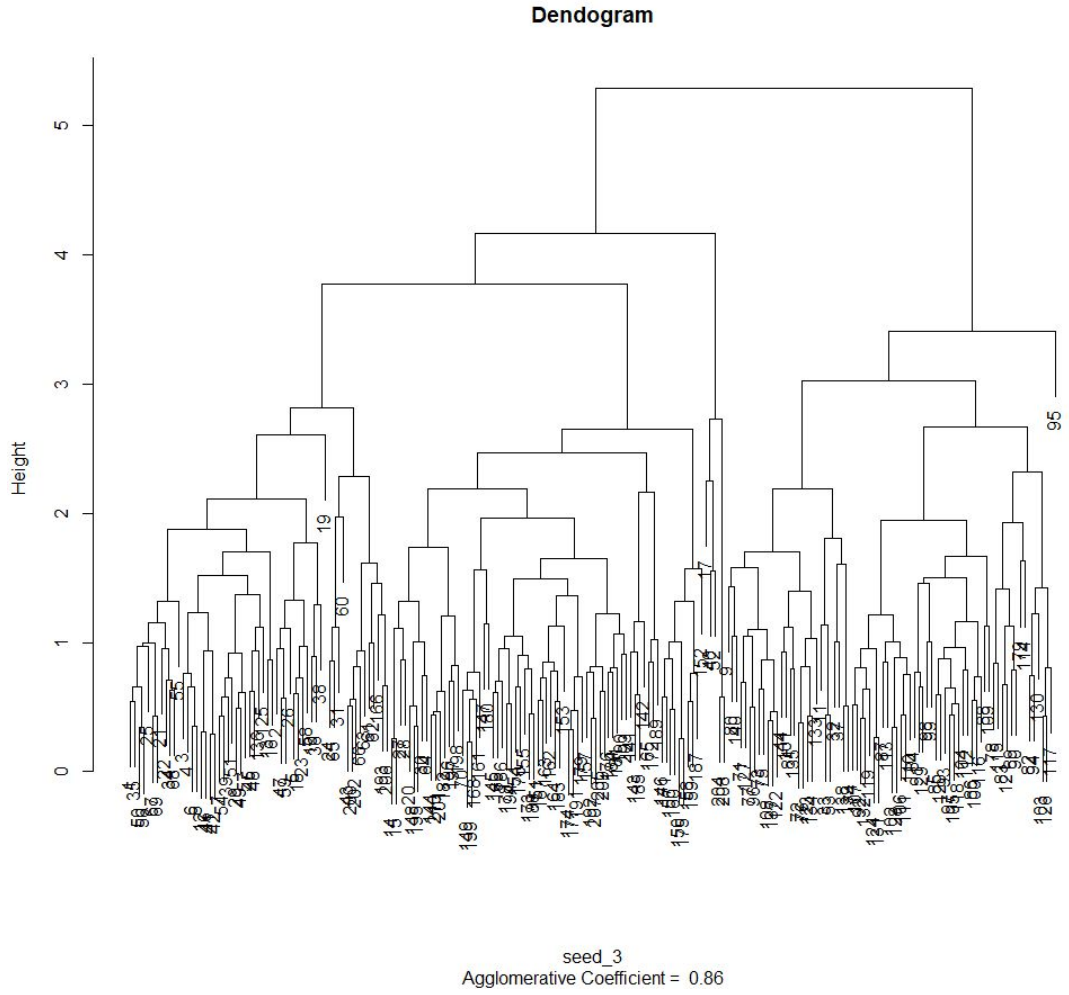
```
# Hierarchical clusterization method
ag1 = agnes(seed_3, metric = "euclidean", stand=TRUE)
```

En este caso, se calcula usando la distancia Euclídea, para seguir la línea de los dos métodos anteriormente aplicados, y con el parámetro **stand** a TRUE, de forma que los datos se estandarizan antes de calcular la distancia.

Después de esto, mostramos el dendograma:

```
windows()
plot(ag1, which.plots=c(2), main="Dendogram")
```

Obteniendo el siguiente resultado:



Como se puede apreciar en el dendograma, si lo cortamos a una altura aproximada de 4, podemos observar cómo obtenemos los 3 clusters de los que hablábamos en la aplicación de los algoritmos anteriores, correspondientes a los 3 tipos de semillas de nuestro dataset. Cortar el dendograma a una altura inferior supondría un mayor número de clusters, pero no aumentaría la información proporcionada por el modelo. Es más, esto sería contraproducente, ya que estaríamos dividiendo un mismo tipo de semilla en dos subgrupos, sin existir diferencias significativas entre ellos como para realizar esa subdivisión. Se aprecia también un outlier en el valor 95, en el extremo derecho de la representación.

6. BIG ML: Clusters

En primer lugar, en los vídeos comunes (Introduction, Sources, Datasets, Supervised and Unsupervised) aprendemos nociones básicas sobre la herramienta BIG ML, sus múltiples utilidades y posibilidades de uso así como su configuración general. Por ejemplo, en los dos primeros vídeos se nos enseña a crear un proyecto, a crear y cargar recursos y a explorar y modificar información de éstos (metadatos). También se nos da una descripción de los distintos tipos de datos soportados por la herramienta y del uso de variables, entre otras muchas funcionalidades.

Por otro lado, en la lección de "Datasets", como tercero de los cuatro vídeos básicos comunes, aprendemos a crear un conjunto de datos para trabajar posteriormente con él. En dicho vídeo se muestran diversas maneras de hacerlo, desde la one-click option, sin tener que configurar nada y confiando en la herramienta a modo de caja negra, hasta la opción "batch", personalizando todos los parámetros y ajustes de entrada al crear el conjunto de datos para tener el control en todo momento sobre las opciones del dataset según nuestros deseos o necesidades. Existen también alternativas asistidas por la propia herramienta, así como la posibilidad de crearlo a partir de otro dataset (quedándonos con una parte del mismo, sólo con algunas instancias, determinadas variables, o realizando modificaciones sobre el original). En esta lección se profundiza en las diversas correcciones o modificaciones que podemos hacer sobre un conjunto de datos ya cargado (a modo de preprocesamiento), se analizan y explican múltiples estadísticas sobre el dataset que vayamos a usar. Por último, en este vídeo se explica la manera de realizar divisiones en conjuntos de entrenamiento y de validación (así como las opciones que proporciona la herramienta al separar los datos en estos dos conjuntos). Además, muestra y explica diversas visualizaciones de los datos que nos pueden dar información muy útil sobre nuestros ellos, a modo de previsualización de nuestro dataset.

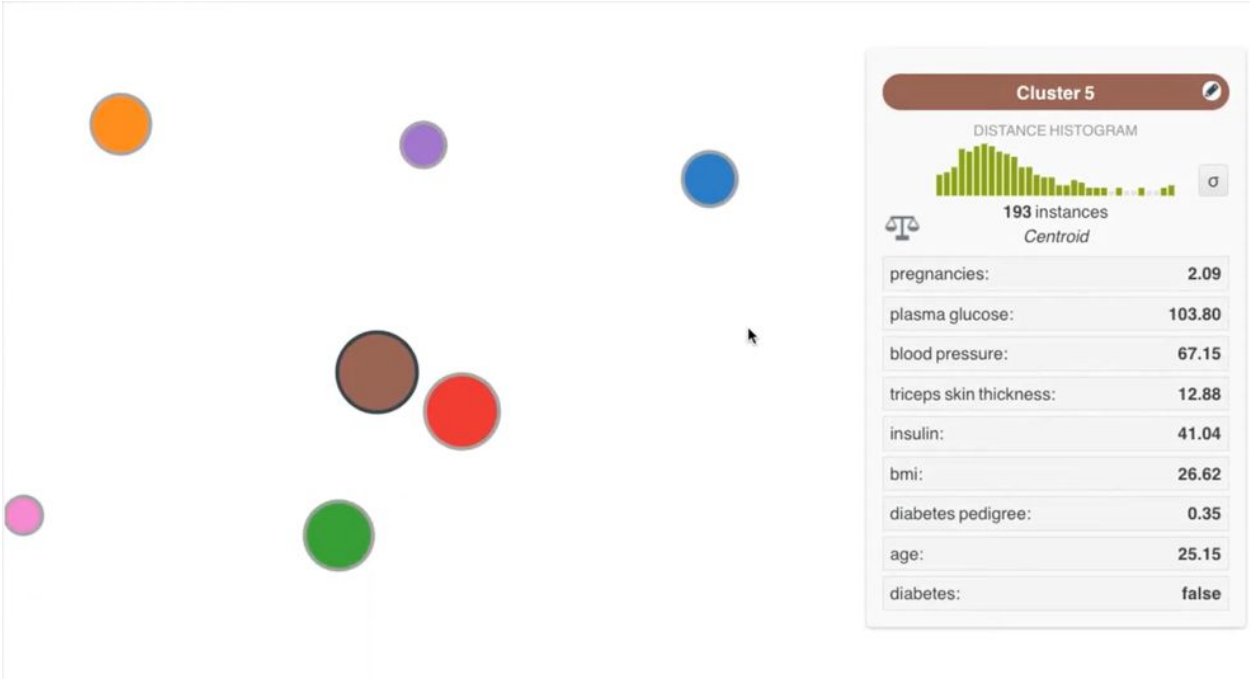
Por último, el vídeo de "Unsupervised and Unsupervised" nos da una rápida a la vez que completa visión teórica con las características de cada tipo de aprendizaje, las diferencias entre ambos tipos y sus respectivas aplicaciones prácticas. Sin embargo, al no tratar específicamente sobre la herramienta BIG ML, y ser una aproximación teórica que no nos resulta nueva, al contar previamente con ese conocimiento por las clases de la propia

asignatura (en las que se profundiza mucho más que en esta breve introducción), no estimamos necesario detenernos demasiado en el contenido de esta lección teórica,

Centrándonos en el vídeo de "Clusters", herramienta que nos ha tocado analizar en profundidad, BIG ML proporciona muy diversos y potentes métodos y alternativas para realizar análisis no supervisados de agrupación.

La primera opción y más básica es la conocida como "one click option", que como su nombre puede hacer intuir, en esencia consiste en dejar que la herramienta realice la clusterización según su propio criterio, utilizando dicha utilidad a modo de "caja negra". Sin embargo, pese a lo simple que pueda parecer esta opción, resulta realmente útil.

Como podemos observar en la primera de las imágenes inferiores, la herramienta nos devuelve una clusterización de la que no tenemos información a priori, ya que el número de clusters ni lo hemos determinado nosotros ni sabemos en base a qué se está fijando un cierto valor de K (7 el caso inferior). En la parte derecha de la imagen inferior podemos apreciar los valores medios del centroide de cada grupo (en este caso el número 5, de color marrón) y comparar las diferencias de esos valores medios de cada variable con los otros clusters. De esta forma, podremos llegar a conclusiones del estilo de "en el cluster X se han agrupado a las instancias que tienen la variable Y con un valor Z".



Adicionalmente, podemos obtener información sobre otras métricas del cluster (valor máximo, mínimo, media, desviación típica o varianza, por ejemplo), podremos analizar su histograma para ver cuán compacto es, o medir la distancia relativa a otros clusters.

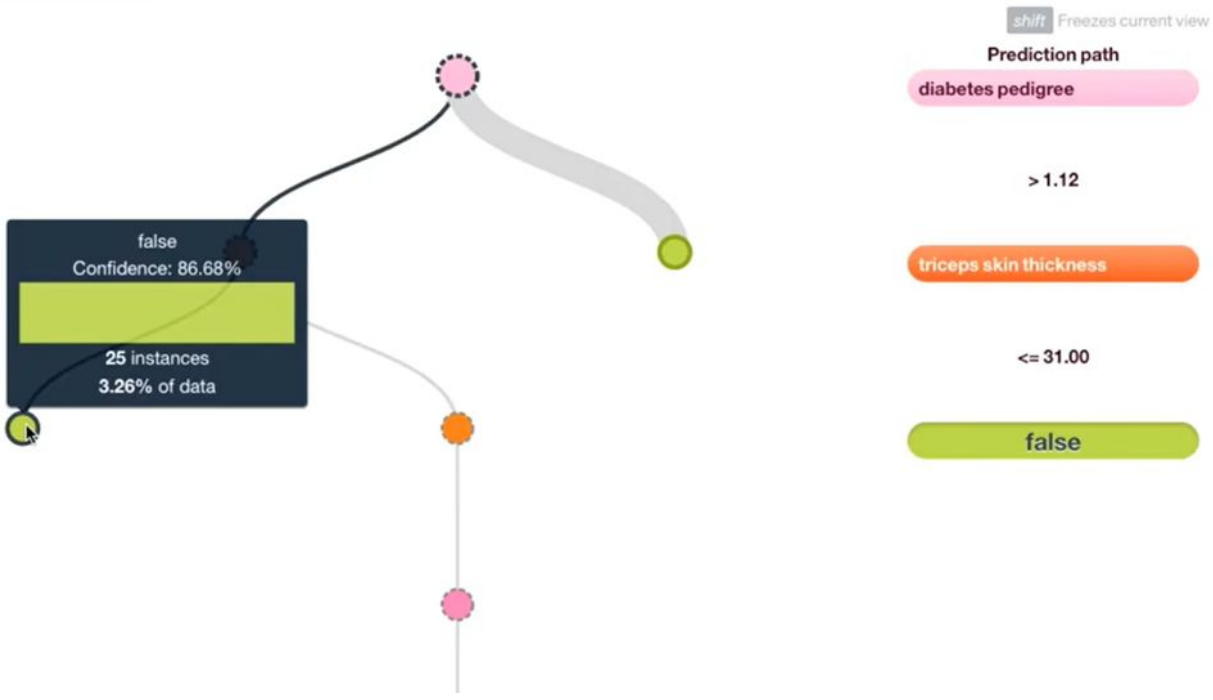
Por último, esta opción nos permite visualizar y exportar un "Cluster Summary Report" en el que se detallan por escrito una gran mayoría de las métricas, atributos y variables antes descritas, recogiendo una amplísima y variada cantidad de información en dicho informe, elaborado con una "one click option" al igual que la clusterización.

Además de la "one click option" anteriormente explicada, la herramienta nos ofrece la posibilidad de configurar ciertos parámetros a la hora de realizar una clusterización algo más personalizada, teniendo nosotros el control de algunos de esos ajustes, en lugar de dejarlo todo en manos de la herramienta como en el caso anterior. En cualquier caso, la salida de los métodos es la misma que en la opción anterior, interpretándose los resultados de la misma forma.

Como podemos apreciar en las dos imágenes siguientes, la configuración de esta opción nos permite escoger entre el algoritmo K-means y G-means. En caso de seleccionar el primero de ellos, deberemos indicar el valor del parámetro K (número de clusters que deseamos que devuelva la salida del método), mientras que si elegimos la segunda opción, deberemos indicar el valor crítico. En este último caso, a diferencia de K-means, no sabremos a priori el número de clusters que retornará, pero con el parámetro del valor crítico podremos ajustarlo implícitamente, ya que a menor critical value, mayor número de clusters y viceversa.

Además de la elección entre los métodos K-means y G-means, y su correspondiente parámetro (K o el valor crítico respectivamente), tendremos la posibilidad de indicar un valor numérico por defecto (para sustituir los valores perdidos de nuestro dataset, NaN, undefined, etc.). Podremos sustituirlos por el valor medio de esa variable, el máximo, el mínimo, la mediana, o el valor cero, según nuestras preferencias o necesidades.

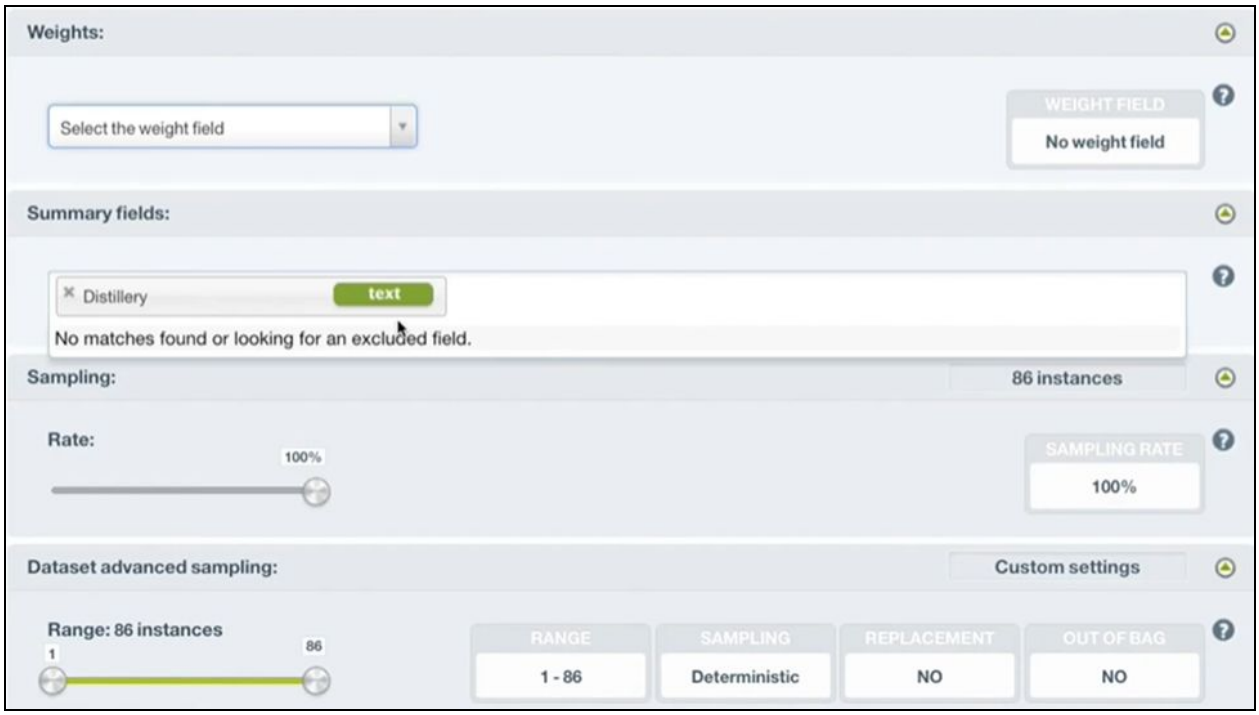
Por último, podremos hacer que la herramienta nos devuelva el modelo del cluster si es que así lo deseamos (última opción a la derecha en las dos imágenes anteriores). Esto nos imprimirá en pantalla una estructura de árbol como la que tenemos en el ejemplo de la siguiente imagen.



La representación del modelo que podemos ver en el ejemplo de la imagen anterior nos permite conocer de manera visual cómo se ha formado cada uno de los clusters que nos devuelve la salida. En este caso, podemos apreciar como en el nodo verde seleccionado hay un total de 25 instancias, en las que se recogen aquellas con un valor de “diabetes pedigree” mayor de 1.12 y un “triceps skin thickness” con un valor mayor o igual a 31.00. Las 25 instancias que forman este cluster de color verde tienen un valor para la clasificación de diabetes de “falso” (no tienen diabetes), con una confianza del 86.68%.

Con esta información, un experto en el dominio del problema podrá sacar conclusiones derivadas de las salidas de la herramienta, interpretando dichos resultados. Una persona que no conozca el contexto del problema, posiblemente no sepa qué significa que el valor de “diabetes pedigree” sea mayor o menor que 1.12, por ejemplo. Sin embargo, un experto en la materia, con esta información, podrá rápidamente identificar qué pacientes son los que forman el nodo verde, ya que sabe las características de las 25 instancias que lo forman.

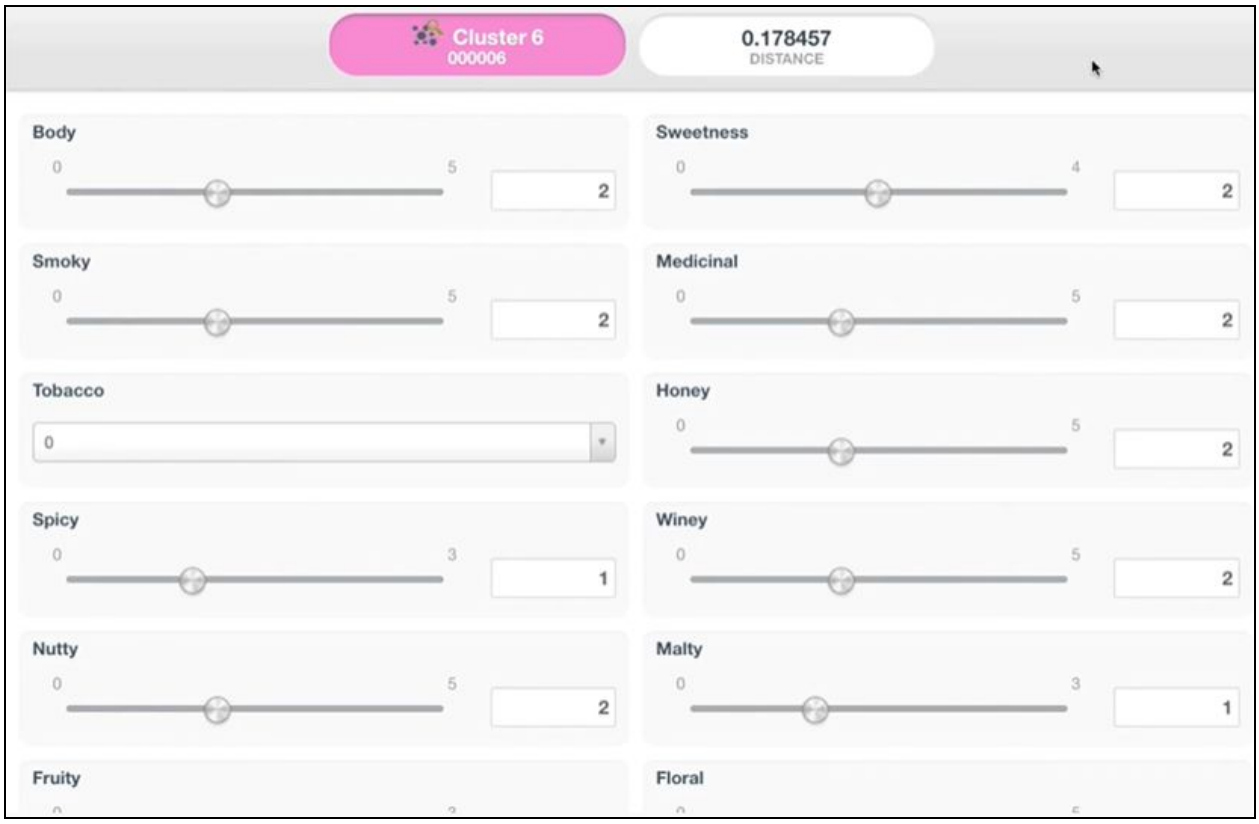
BIG ML nos ofrece una opción aún más personalizable, entrando de lleno en la configuración del dataset y la clusterización a realizar, tal y como podemos ver en las dos siguientes imágenes.



Con esta configuración, tenemos poder de decisión más sobre el uso de los datos que sobre los propios métodos de clusterización; esto segundo se nos da en la opción anterior (al elegir entre K-means y G-means y sus parámetros). En este caso, como vemos en las dos imágenes anteriores, los parámetros a ajustar son el escalado de los valores, la asignación de peso a una determinada variable (si queremos que tenga más importancia que las demás), o la separación en conjuntos de training y test (para que no use el dataset al completo para el entrenamiento). Como podemos ver, son opciones más parecidas a un preprocesado que ajustes propios de los métodos de clusterización a realizar.

Por último, podemos utilizar la interfaz llamada “BatchCentroids”, que nos ofrece la posibilidad de “jugar” de una manera muy visual con los valores de cada una de las variables, como se muestra en la imagen inferior. Así, podremos ir variando los distintos valores de cada atributo del dataset (moviendo a derecha e izquierda la correspondiente barra horizontal) y automáticamente la herramienta nos dirá a qué cluster correspondería una instancia con los valores que acabamos de fijar. De esta manera, podemos conocer

qué tipo de datos entra en cada cluster, en base a los propios valores de las variables, antes incluso de realizar la propia clusterización.



En conclusión, hemos podido comprobar que BIG ML es una herramienta tremendamente potente, que proporciona una gran cantidad de información (haciendo mención especial a los métodos de clusterización que nosotros hemos estudiado), pero que no por ello es una herramienta compleja. Precisamente en su sencillez y en su interfaz visual y asistida es donde reside la gran utilidad de la aplicación, ya que permite realizar técnicas muy potentes sin necesidad de entrar en código ni programación, haciéndolo todo más fácil y cómodo al usuario, sin que ello suponga obtener una respuesta simple o con poca información. Proporciona la información y el conocimiento propio de un análisis tradicional, pero con las facilidades que da el uso de la interfaz asistida de BIG ML.

7. Bibliografía

Knime:

- https://forum.knime.com/t/grouping-nodes-in-knime/6241
- https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.groupby.GroupByNodeFactory
- https://hub.knime.com/knime/extensions/org.knime.features.base/latest/org.knime.base.node.preproc.normalize3.Normalizer3NodeFactory

Caret package:

- https://topepo.github.io/caret/index.html

RPart:

- https://www.rdocumentation.org/packages/rpart/versions/4.1-15/topics/rpart
- https://cran.r-project.org/web/packages/rpart/rpart.pdf
- https://rpubs.com/jboscomendoza/arboles_decision_clasificacion

CTree2:

- https://github.com/topepo/caret/blob/master/RegressionTests/Code/ctree2.R

Naive-Bayes:

- https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/naiveBayes
- https://rpubs.com/mao045/485540

Kmeans:

- https://en.wikipedia.org/wiki/K-means_clustering

Kmedioids:

- http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html

Clustering:

- https://rpubs.com/Joaquin_AR/310338#:~:text=K%2Dmedoids%20es%20un%20m%C3%A9todo,valor%20preestablecido%20por%20el%20analista.

Agnes:

- http://rstudio-pubs-static.s3.amazonaws.com/387689_5e8f2e3408eb4e52b3e2a4cfbc400553.html#:~:text=Tambi%C3%A9n%20se%20lo%20conoce%20como,similares%20se%20fusionan%20en%20uno.

BIG ML:

- https://bigml.com/
- https://bigml.com/education/videos
- https://www.youtube.com/watch?list=PL1bKyu9GtNYHAk0PUojkLYZzASoYVcsTQ&v=6nIsu82hCig&feature=youtu.be&ab_channel=bigmlcom