Tratamiento Inteligente de Datos

Análisis de sentimientos sobre App Google Store

Autor
María Matilde Cabrera González

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Enero de 2020

Índice:

Introducción

Si bien muchos conjuntos de datos públicos (en Kaggle y similares) proporcionan datos de Apple App Store, no hay muchos conjuntos de datos de contrapartida disponibles para las aplicaciones de Google Play Store en cualquier lugar de la web. Al profundizar más, descubrimos que la página de la tienda de aplicaciones de iTunes implementa una estructura similar a un apéndice bien indexada para permitir un raspado web simple y fácil. Por otro lado, Google Play Store utiliza técnicas sofisticadas de hoy en día (como la carga dinámica de páginas) usando JQuery, lo que hace que el raspado sea más desafiante

Dataset

Enlaces a los sitios donde se pueden descargar los dataset:
https://www.kaggle.com/lava18/google-play-store-apps
https://www.kaggle.com/lava18/google-play-store-apps#googleplaystore_user_reviews.csv

Estos son los datos raspados en la web de aplicaciones de 10k Play Store para analizar el mercado de Android.

Tenemos dos archivos “googleplaystore.csv” y “googleplaystore_user_reviews.csv”.

El primero de ellos tiene 13 columnas:

App: Nombre de la aplicación.
Category: Categoría a la que pertenece la aplicación.
Rating: Calificación general del usuario de la aplicación (como cuando se raspa).
Reviews: Número de revisiones de usuarios para la aplicación (como cuando se raspa).
Size: Tamaño de la aplicación (como cuando se raspa).
Installs: Número de descargas/instalaciones de usuarios para la aplicación (como cuando se raspa).
Type: Pagado o Gratis.
Price: Precio de la aplicación (como cuando se raspa).
Content Rating: Grupo de edad al que se dirige la aplicación: niños / adultos mayores de 21 años / adultos.
Genres: Una aplicación puede pertenecer a múltiples géneros (aparte de su categoría principal). Por ejemplo, un juego familiar musical pertenece a los géneros Música, Juego, Familia.
Last Updated: Fecha en que la aplicación se actualizó por última vez en Play Store (como cuando se raspó).

Current Ver: Versión actual de la aplicación disponible en Play Store (como cuando se raspa).
Android Ver: Versión mínima requerida de Android (como cuando se raspa).

El segundo documentos de los review tiene 5 columnas:

App: Nombre de la aplicación.
Translated_Review: Revisión del usuario (pre procesada y traducida al inglés).
Sentiment: Positivo/Negativo/Neutro (preprocesado).
Sentiment_Polarity: Puntuación de polaridad de sentimiento.
Sentiment_Subjectivity: Puntuación de subjetividad sentimental.

Problema a resolver

Entrenar modelos para clasificar el sentimiento que denota un texto y detectar qué atributos tienen relación con un sentimiento positivo/negativo/neutro para analizar el mercado de Android. P

Análisis exploratorio de los datos

Preprocesamiento

Tenemos dos archivos, vamos a unirlos, primero trataremos los datos del fichero “googleplaystore_user_reviews” y luego añadiremos las comunas del otro fichero donde los nombres de la app sean iguales. Empezamos:

Contar número de nulos por columna

```
> sapply(googleplaystore_user_reviews, function(x) sum(is.na(x)))
                  App      Translated_Review              Sentiment
                    0                      5                      0
    Sentiment_Polarity Sentiment_Subjectivity
                 26863                  26863
```

Eliminar todas las filas que contengan algún valor nulo

```
> delete.na <- function(df, n=0) {
+     df[rowSums(is.na(df)) <= n,]
+ }
> googleplaystore_user_reviews <- na.omit(googleplaystore_user_reviews)
```

“googleplaystore_user_reviews” pasa de 64295 datos a 37427.

Instalar paquete “dplyr”:

```
> install.packages("dplyr")
```

Cargar el paquete:

```
> library("dplyr")
```

Queremos saber la cantidad de sentimientos positivos, negativos o neutros.

```
> googleplaystore_user_reviews %>% group_by(Sentiment) %>% summarise(n = n())
# A tibble: 3 x 2
  Sentiment     n
  <chr>     <int>
1 Negative   8271
2 Neutral    5158
3 Positive  23998
```

Vamos a dejar 4500 datos en proporción a los datos actuales, serían:

negativo	1500
neutral	1500
positivo	1500

Esto lo hacemos así para tener un balanceo equitativo de los datos.

Dividimos nuestro dataset en 3, filtrando por sentimientos:

```
> positive <- googleplaystore_user_reviews %>% filter(Sentiment=="Positive")
> negative <- googleplaystore_user_reviews %>% filter(Sentiment=="Negative")
> neutral <- googleplaystore_user_reviews %>% filter(Sentiment=="Neutral")
```

Para positive:

```
> filas.random <- sample(1:23998, 1500, replace= F)
> positive2 <- as.data.frame(positive[filas.random,])
```

Para negative:

```
> filas.random <- sample(1:8271, 1500, replace= F)
> negative2 <- as.data.frame(negative[filas.random,])
```

Para neutral:

```
> filas.random <- sample(1:5158, 1500, replace= F)
> neutral2 <- as.data.frame(neutral[filas.random,])
```

Ahora unimos los review a googleplaystore y quitamos los duplicados

```
> unidos = rbind(positive2,negative2,neutral2)
> unidos <- unidos[!duplicated(unidos), ]
> googleplaystore_user_reviews = merge(unidos, datosplaystore)
```

Quitamos repetidos

```
datosprogoogleplaystore <-
datosprogoogleplaystore[!duplicated(datosprogoogleplaystore), ]
```

Descargamos el fichero:

```
write.csv(datosprogoogleplaystore, file="datosprogoogleplaystore.csv")
```

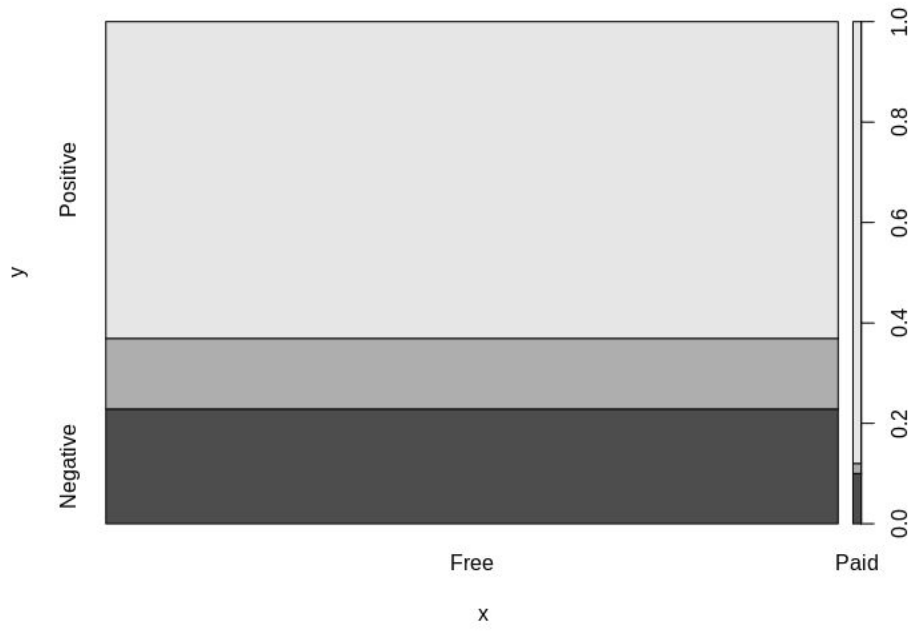
Análisis

Vamos a ver gráficamente qué variables tienen relación entre el sentimiento y el resto de columnas:
Factorizamos las columnas:

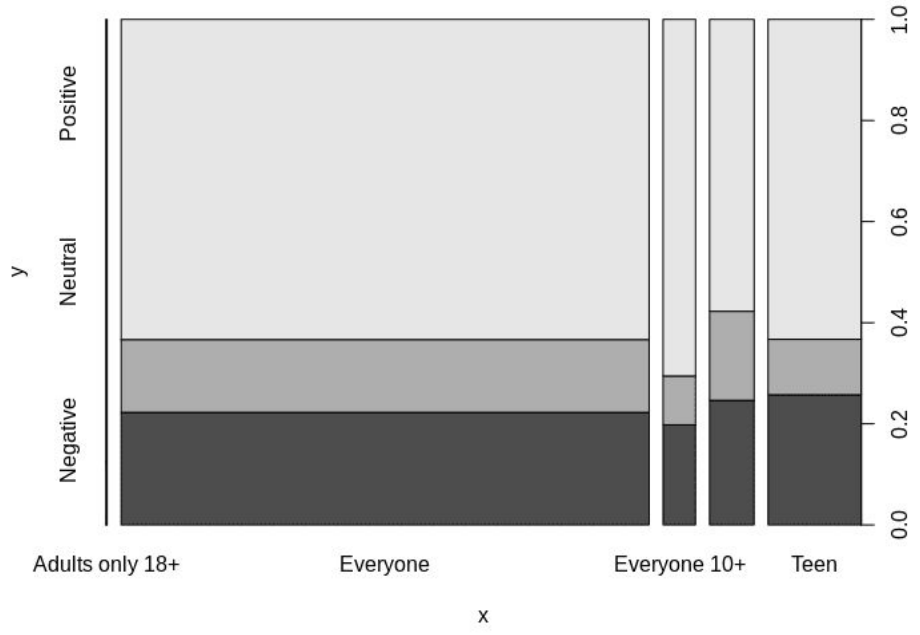
```
> Sentiment = as.factor(datos$Sentiment)
> Category = as.factor(datos$Category)
> Review = as.factor(datos$Translated_Review)
> Type = as.factor(datos$Type)
> Price = as.factor(datos$Price)
> Genres = as.factor(datos$Genres)
> Installs = as.factor(datos$Installs)
> Rating = as.factor(datos$Content.Rating)
> App = as.factor(datos$App)
```

Vemos visualmente:

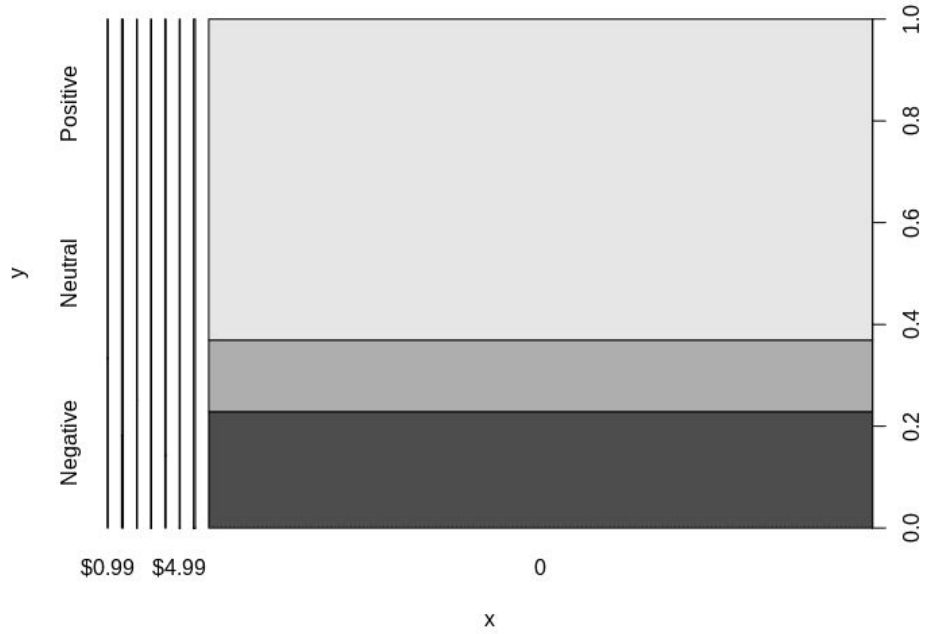
```
> plot(Type,Sentiment)
```



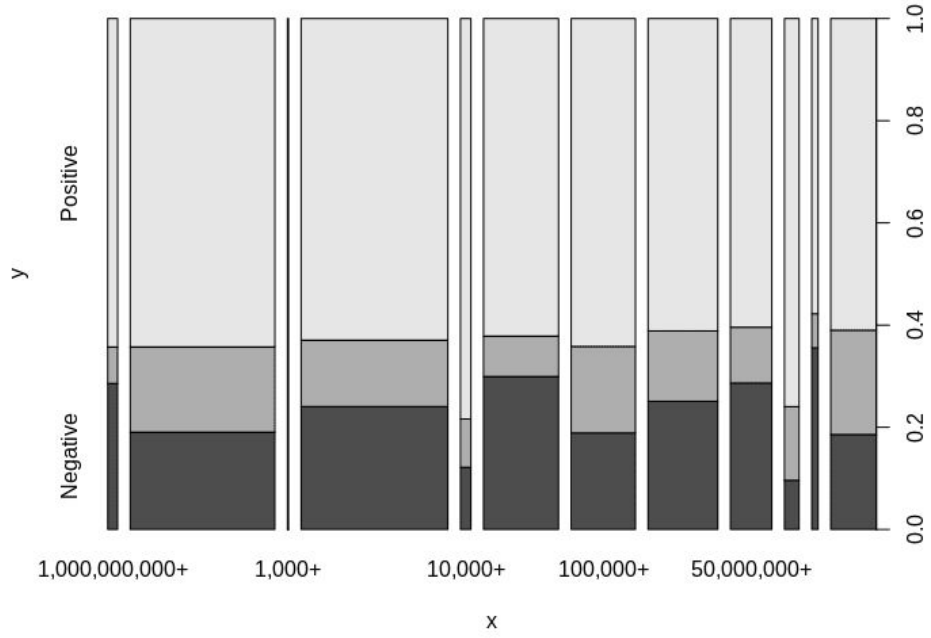

```
> plot(Review,Sentiment)
> plot(Rating,Sentiment)
```



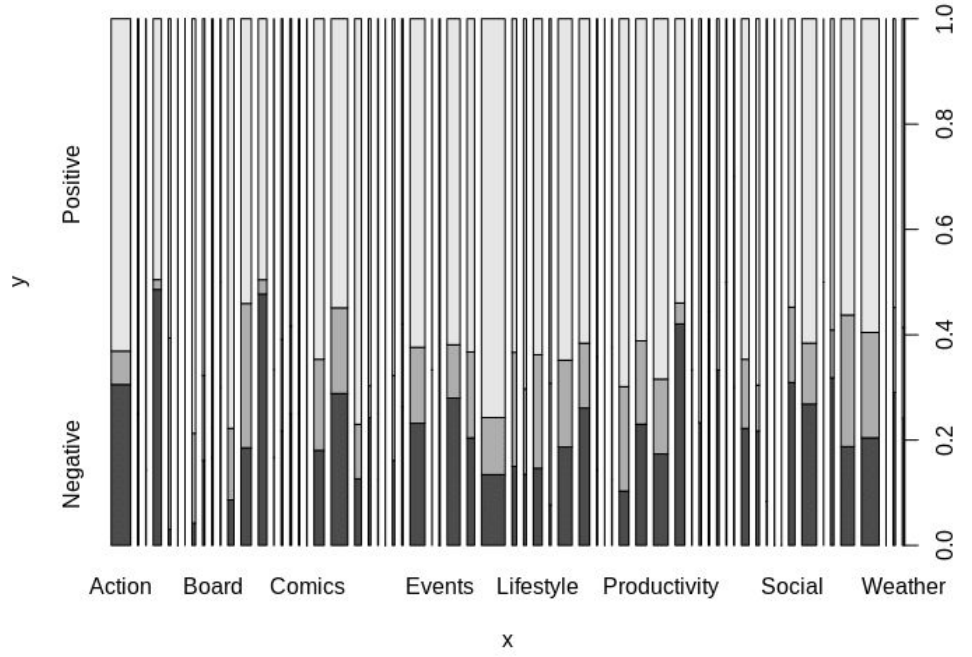
```
> plot(Price,Sentiment)
```



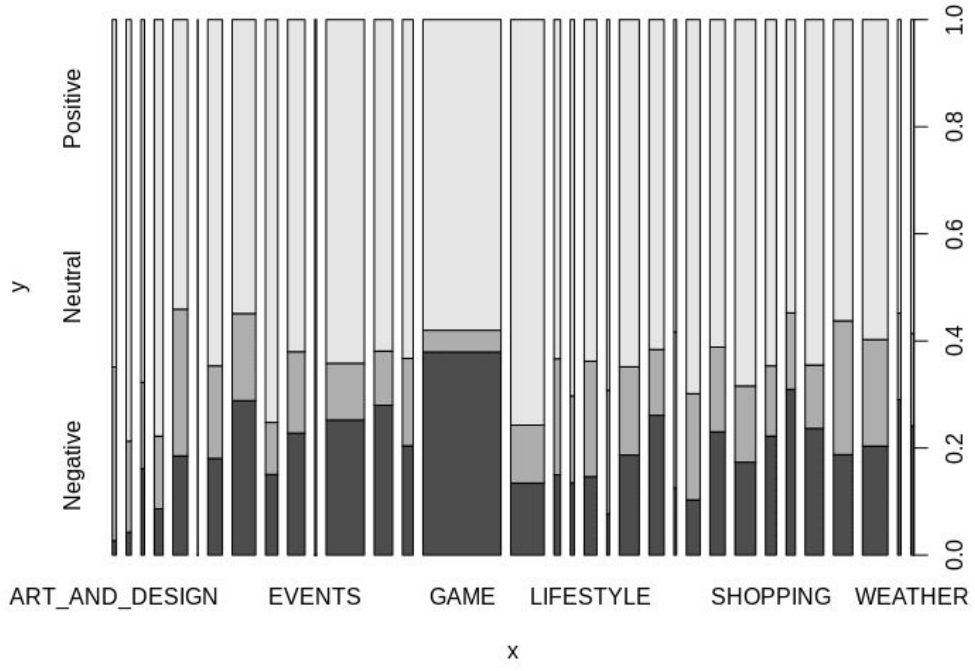

```
> plot(Installs,Sentiment)
```



```
> plot(Genres,Sentiment)
```

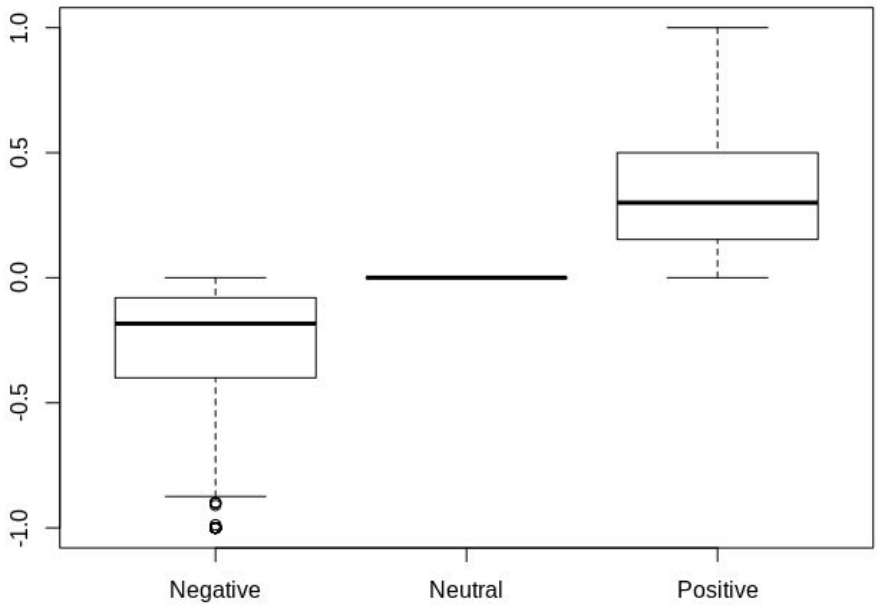



```
> plot(Category,Sentiment)
```

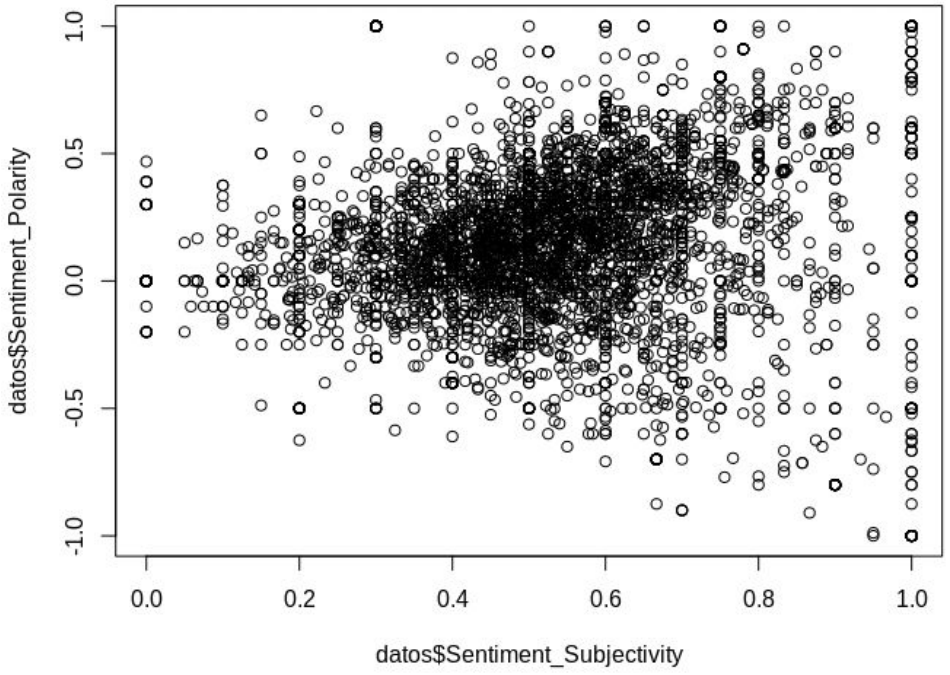



```
> plot(App,Sentiment)
```

```
> plot(Sentiment,datos2$Sentiment_Polarity)
```




```
> plot(datos$Sentiment_Subjectivity,datos$Sentiment_Polarity)
```



Después de un primer vistazo no observamos que ninguna columna sea objetiva para el análisis de sentimientos, excepto su polaridad y subjetividad que como es obvio no lo podemos usar.

```
> summary(datos2)
   Sentiment     Sentiment_Polarity Sentiment_Subjectivity    Category         Rating         Reviews
Min.   :1.000   Min.   :-1.0000    Min.   :0.0000         Min.   : 1.00   Min.   :2.600   Min.   :      46
1st Qu.:2.000   1st Qu.: 0.0000    1st Qu.:0.3500         1st Qu.:12.00   1st Qu.:4.100   1st Qu.:   10247
Median :3.000   Median : 0.1354    Median :0.5030         Median :15.00   Median :4.300   Median :   63624
Mean   :2.406   Mean   : 0.1657    Mean   :0.4882         Mean   :17.39   Mean   :4.294   Mean   : 1334050
3rd Qu.:3.000   3rd Qu.: 0.3806    3rd Qu.:0.6500         3rd Qu.:25.00   3rd Qu.:4.500   3rd Qu.:  471036
Max.   :3.000   Max.   : 1.0000    Max.   :1.0000         Max.   :33.00   Max.   :4.900   Max.   :78158306

      Size           Installs           Type           Price       Content.Rating      Genres
Min.   :  1.00   Min.   : 1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   : 1.00
1st Qu.: 34.00   1st Qu.: 4.000   1st Qu.:1.000   1st Qu.:8.000   1st Qu.:2.000   1st Qu.:21.00
Median :103.00   Median : 6.000   Median :1.000   Median :8.000   Median :2.000   Median :33.00
Mean   : 96.96   Mean   : 5.615   Mean   :1.011   Mean   :7.957   Mean   :2.572   Mean   :34.34
3rd Qu.:163.00   3rd Qu.: 8.000   3rd Qu.:1.000   3rd Qu.:8.000   3rd Qu.:2.000   3rd Qu.:46.00
Max.   :163.00   Max.   :12.000   Max.   :2.000   Max.   :8.000   Max.   :5.000   Max.   :66.00
```

Después de estos resultados definimos que ninguna columna nos sirve para el estudio de sentimientos, nos disponemos a realizar el estudio del texto seleccionado en la columna “Translated_Review” .

Preprocesamiento y análisis semántico.

Como vamos a usar una única columna, obtenemos 1500 datos de cada sentimiento de la misma forma que en el preprocesamiento anterior.

```
Trans_review = datosprocesados$Translated_Review
Trans_review = as.matrix(Trans_review)
```

Limpiamos el texto para poder trabajar con el mismo.

```
# Elimina numeros
Trans_review <- gsub("[[:digit:]]", "", Trans_review)
# Elimina RT
Trans_review <- gsub("RT","",Trans_review)
# Elimina espacios en blanco múltiples
Trans_review <- gsub("[\\s]+", "", Trans_review)
# Elimina signos como @
Trans_review <- gsub("([^[:alpha:] ]+|&amp;|&am;|\\n+)", "", Trans_review)
# Eliminacion de signos de puntuación
Trans_review <- gsub("[[:punct:]]", "", Trans_review)
# Pasar de mayuscula a minuscula
Trans_review <- tolower(Trans_review)
```

Para trabajar con documentos, necesitamos la entidad corpus que gestiona documentos de texto en lenguaje natural de manera genérica, para usar Corpus necesitamos el paquete “™”.

```
corpus.tmp <- Corpus(VectorSource(Trans_review))
```

Generamos la matriz de documentos-términos o matriz de documentos a plazo con "`dtm <- DocumentTermMatrix(corpus.tmp)`", además se suprimen términos que no tengan significado, vamos a eliminar los términos con longitud menos de 3 letras y mayores de 12 porque entendemos que son caracteres sin sentido. Como parámetros tenemos Corpus.tmp que es una matriz o un vector de frecuencia de término.

```
dtm <- DocumentTermMatrix(corpus.tmp, list(wordLengths= c(3,10)))
```

Para filtrar la matriz de documentos-términos por frecuencias usamos el siguiente comando:

```
dtm <- DocumentTermMatrix(corpus, list(bounds= list(global= c(5,Inf))))
```

Vamos a eliminar los términos que aparecen en muy pocos documentos, para ello inspeccionamos "dtm".

```
tf <- as.matrix(dtm)
> inspect(dtm)
<<DocumentTermMatrix (documents: 4500, terms: 1715)>>
Non-/sparse entries: 48788/7668712
Sparsity           : 99%
Maximal term length: 13
Weighting          : term frequency (tf)
Sample             :
      Terms
Docs   app cant game get good like the thi time work
  1247   0    0    1   2    0    1   1   0    0    1
  1855   2    0    0   1    1    0   2   1    0    0
  198    0    0    0   0    0    0   3   2    0    1
  2363   1    1    0   0    0    3   0   0    1    3
  2722   0    0    0   1    0    0   0   4    0    0
  2787   1    0    0   3    0    1   0   0    2    0
  2817   0    0    5   3    0    1   0   1    3    0
  620    0    0    0   0    0    0   3   2    0    1
  689    0    0    4   1    0    1   2   0    0    0
  733    0    0    2   1    0    0   1   0    2    0
```

Como el umbral de escasez es 0,99, se toman los términos que aparecen en más del 1 % de documentos.

```
dtm <- removeSparseTerms(dtm, sparse= 0.98)
> inspect(dtm)
<<DocumentTermMatrix (documents: 4500, terms: 6997)>>
Non-/sparse entries: 56817/31429683
Sparsity           : 100%
```

```
Maximal term length: 12
Weighting          : term frequency (tf)
Sample             :
      Terms
Docs   app cant game get good like the thi time work
  1247   0    0    1   2    0    1   1   0    0    1
  1855   2    0    0   1    1    0   2   1    0    0
  1859   0    0    5   1    0    0   0   1    2    0
  2495   1    0    0   0    0    1   1   0    0    0
  2722   0    0    0   1    0    0   0   4    0    0
  2787   1    0    0   3    0    1   0   0    2    0
  2817   0    0    5   3    0    1   0   1    3    0
  670    0    0    0   0    0    0   4   0    0    0
  689    0    0    4   1    0    1   2   0    0    0
  733    0    0    2   1    0    0   1   0    2    0
```

Con esto damos por terminado esta sección.

Agrupamiento (cluster)

Es una técnica de aprendizaje no supervisado que permite inferir modelos para extraer conocimiento de conjuntos de datos desconocidos hasta el momento.

Preprocesamiento

Ya tenemos el análisis del cluster y la reducción de dimensiones realizado en el apartado análisis.

Para estudiar la importancia de los términos de un documento en particular, en lugar de utilizar la frecuencia de cada uno de los términos directamente, se pueden utilizar diferentes ponderaciones (TF-IDF), las cuales se calculan como el producto de dos medidas, la frecuencia de aparición del término (tf) y la frecuencia inversa del documento (idf).

```
# vector de pesos en idf
idf <- log(nrow(tf)/(colSums(tf!=0))+1)

# Matriz tf-idf
dtidf <- t(t(tf)*idf)
```

Agrupamiento Jerárquico

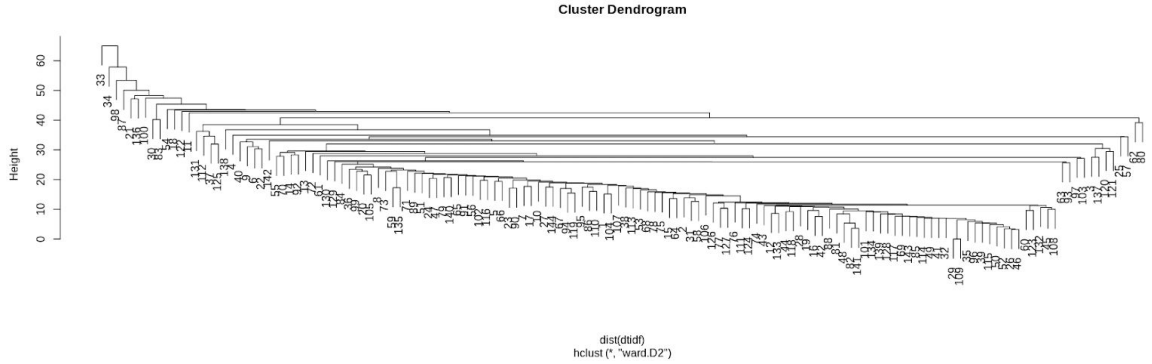
El agrupamiento jerárquico es un método para agrupar datos basándose en las distancias entre cada uno y buscando que los datos que están dentro del mismo clúster sean lo más similar posible.

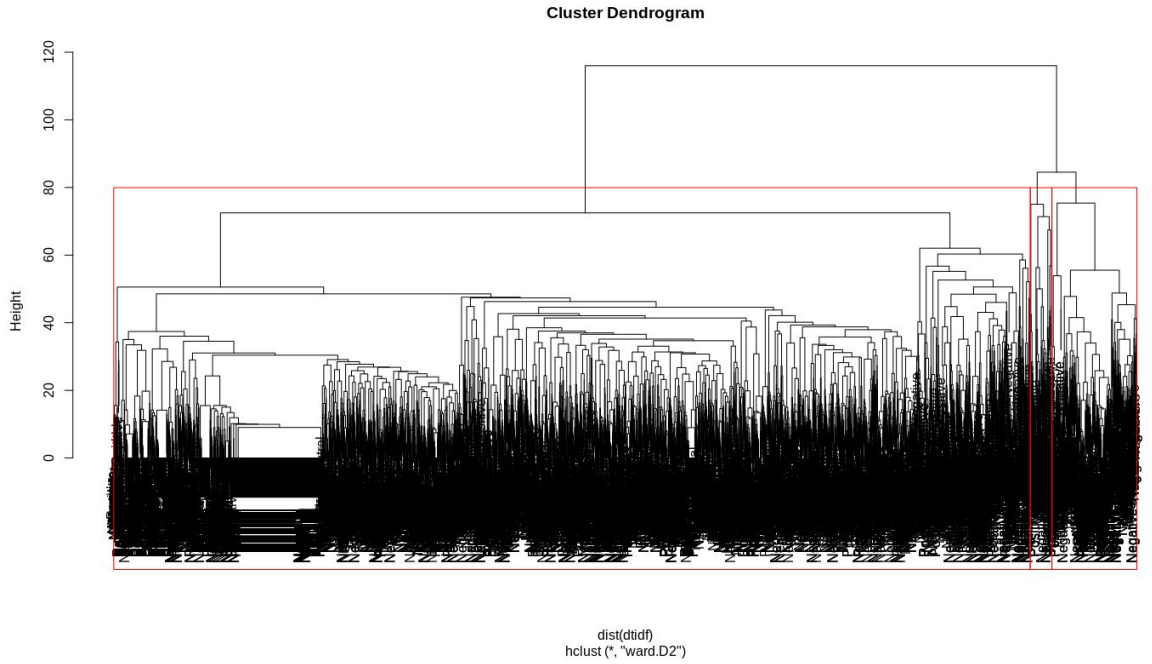
En este como en todos los algoritmos de aprendizaje no supervisado, los datos no están etiquetados previamente y solo cuenta con variables independientes, los algoritmos buscan patrones en los datos sin hacer una predicción a priori. Esto nos permite descubrir y analizar asociaciones en los datos que con otras técnicas no veríamos.

Calculamos la distancia euclídea de la matriz inversa de documento-termino.

```
> hc=hclust(dist(dtidf),method="ward.D2")
> hc

Call:
hclust(d = dist(dtidf), method = "ward.D2")

Cluster method   : ward.D2
Distance         : euclidean
Number of objects: 4500
```

Dibujamos el dendrograma y cortamos por tres



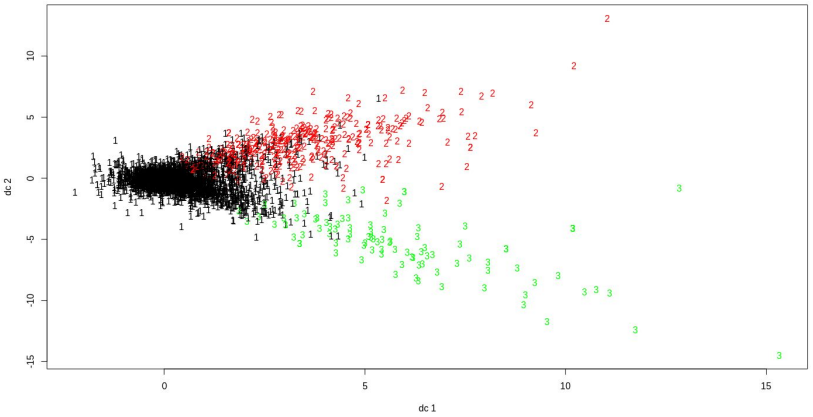


Para generar la variable de agrupamiento, definimos k=3, la función cutree cortará el árbol de palabras en 3 grupos, el que deseamos ya que queremos diferenciar entre sentimientos positivos, negativos y neutros.

```
> group=cutree(hc,k=3)
```

Las medidas de bondad de agrupamiento son el coeficiente de silueta

```
> plotcluster(dtidf,group)
```

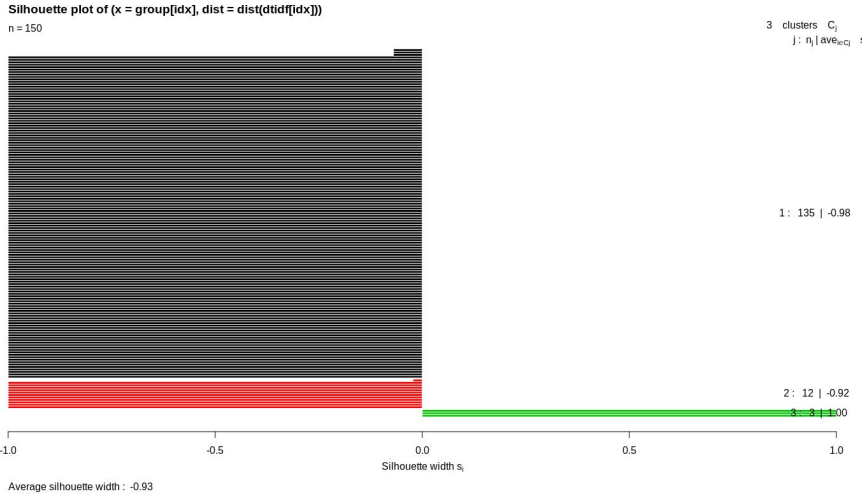


Restringimos el análisis de bondad a 150 valores para poder dibujar, ya que si pongo más valores no se distingue nada en la imagen.

```
idx=sample(1:dim(dtidf)[1],150)
```

En clustering, construimos distintas particiones de los datos y los evaluamos de acuerdo a algún criterio, en nuestro caso de particionamiento. Como medida de rendimiento usamos el coeficiente de silueta, el cual mide cómo de similares son los objetos de un mismo cluster comparado con otros clusters.

```
> shi= silhouette(group[idx],dist(dtidf[idx]))
> plot(shi,col=1:3)
```



Otras medidas de bondad

```
>
cluster.stats(dist(dtidf),group,alt.clustering=as.integer(user_reviews$Sentiment))
```

```
$n
[1] 4500

$cluster.number
[1] 3

$cluster.size
[1] 4031  374   95

$min.cluster.size
[1] 95

$noisen
[1] 0

$diameter
[1] 47.33985 48.54383 55.00501

$average.distance
[1] 12.18070 18.32615 26.42699

$median.distance
[1] 11.77440 18.14875 24.64130

$separation
[1] 2.011540 2.011540 7.395038

$average.toother
[1] 17.15241 16.26942 21.51034

$separation.matrix
         [,1]     [,2]     [,3]
[1,] 0.000000 2.011540 7.395038
[2,] 2.011540 0.000000 9.474604
[3,] 7.395038 9.474604 0.000000

$ave.between.matrix
         [,1]     [,2]     [,3]
[1,]  0.00000 16.09567 21.31260
[2,] 16.09567  0.00000 23.64167
[3,] 21.31260 23.64167  0.00000

$average.between
[1] 17.27211

$average.within
[1] 12.99221

$n.between
```

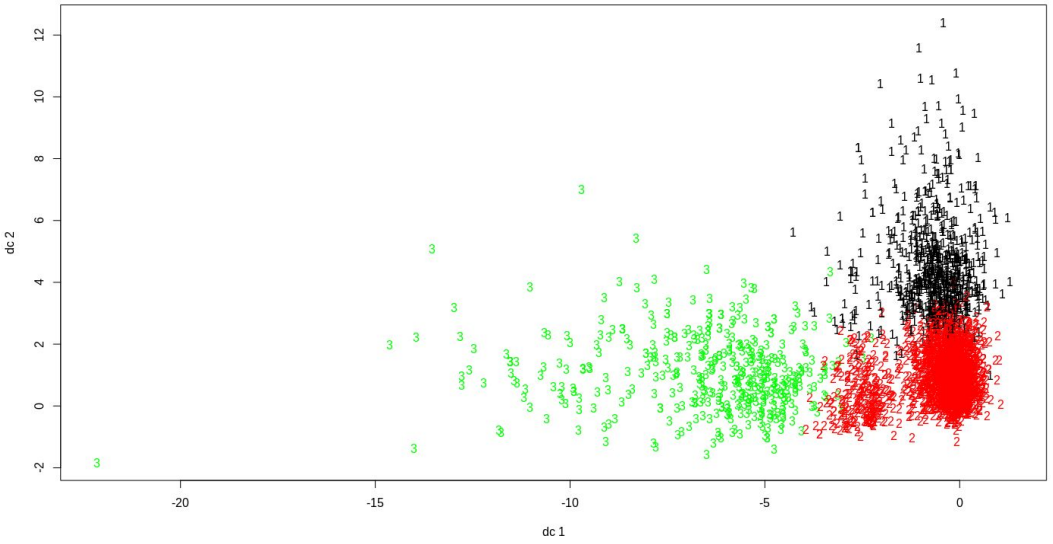
```
[1] 1926069

$n.within
[1] 8196681

$max.diameter
[1] 55.00501

$min.separation
[1] 2.01154

$within.cluster.ss
[1] 457031.8

$clus.avg.silwidths
         1          2          3
 0.2565504 -0.1364451 -0.2081349

$avg.silwidth
[1] 0.214078

$pearsongamma
[1] 0.3409024

$dunn
[1] 0.03657012

$dunn2
[1] 0.609062

$entropy
[1] 0.3867829

$wb.ratio
[1] 0.7522071

$ch
[1] 50.67523

$cwidegap
[1] 33.74080 35.23108 36.31160

$widestgap
[1] 36.3116

$sindex
[1] 2.176919

$corrected.rand
[1] 0.008717152
```

```
$vi
[1] 1.42798
```

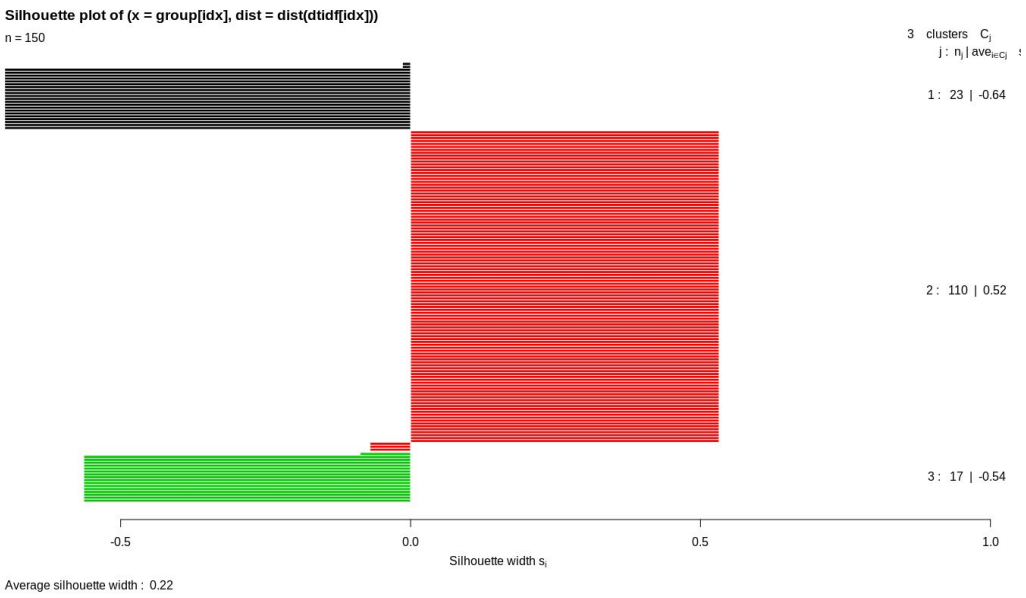
k-medias

En este método cada cluster se representa por el centro del cluster. Necesitamos como parámetro de entrada el número de clusters que deseamos, en nuestro caso 3. K-medias es un algoritmo iterativo en el que las instancias se van moviendo entre clusters hasta que se alcanza el conjunto de clusters deseado.

```
> kmeans.result=kmeans(dtidf,3)
> kmeans.result
K-means clustering with 3 clusters of sizes 570, 41, 3889
```



Vemos la silueta:



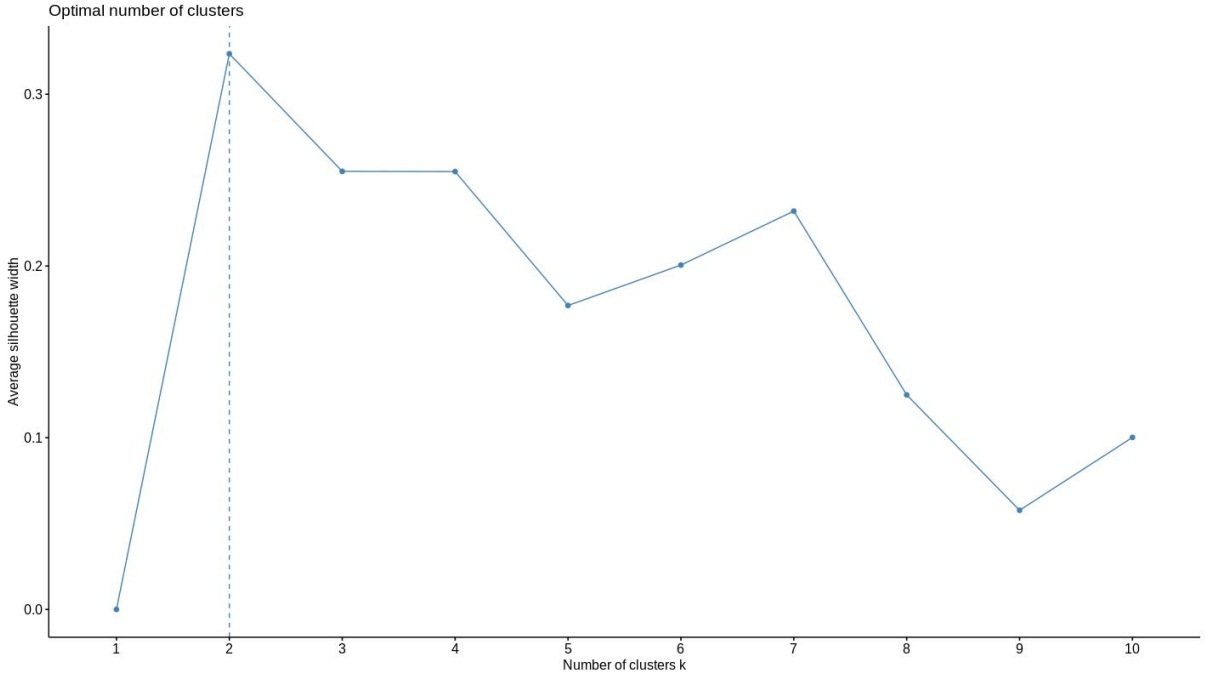

```
> cluster.stats(dist(dtidf),group,alt.clustering=as.integer(user_reviews$Sentiment))
$n
[1] 4500

$cluster.number
[1] 3

$cluster.size
[1] 3333  752  415

$min.cluster.size
[1] 415

$noisen
[1] 0

$diameter
[1] 26.97185 55.00501 53.64453

$average.distance
[1]  9.586501 22.023320 19.687895

$median.distance
[1]  9.641939 20.915774 19.197616

$separation
[1] 2.011540 6.054261 2.011540

$average.toother
[1] 16.90212 17.71793 17.24636
```

```
$separation.matrix
         [,1]     [,2]     [,3]
[1,] 0.000000 6.054261 2.011540
[2,] 6.054261 0.000000 7.399131
[3,] 2.011540 7.399131 0.000000

$ave.between.matrix
        [,1]     [,2]     [,3]
[1,]  0.0000 17.24440 16.28190
[2,] 17.2444  0.00000 21.52099
[3,] 16.2819 21.52099  0.00000

$average.between
[1] 17.24519

$average.within
[1] 12.5964

$n.between
[1] 4201691

$n.within
[1] 5921059

$max.diameter
[1] 55.00501

$min.separation
[1] 2.01154

$within.cluster.ss
[1] 449848

$clus.avg.silwidths
         1          2          3
 0.4186353 -0.2263922 -0.1860463

$avg.silwidth
[1] 0.255079

$pearsongamma
[1] 0.5884993

$dunn
[1] 0.03657012

$dunn2
[1] 0.7393028

$entropy
[1] 0.7411461
```

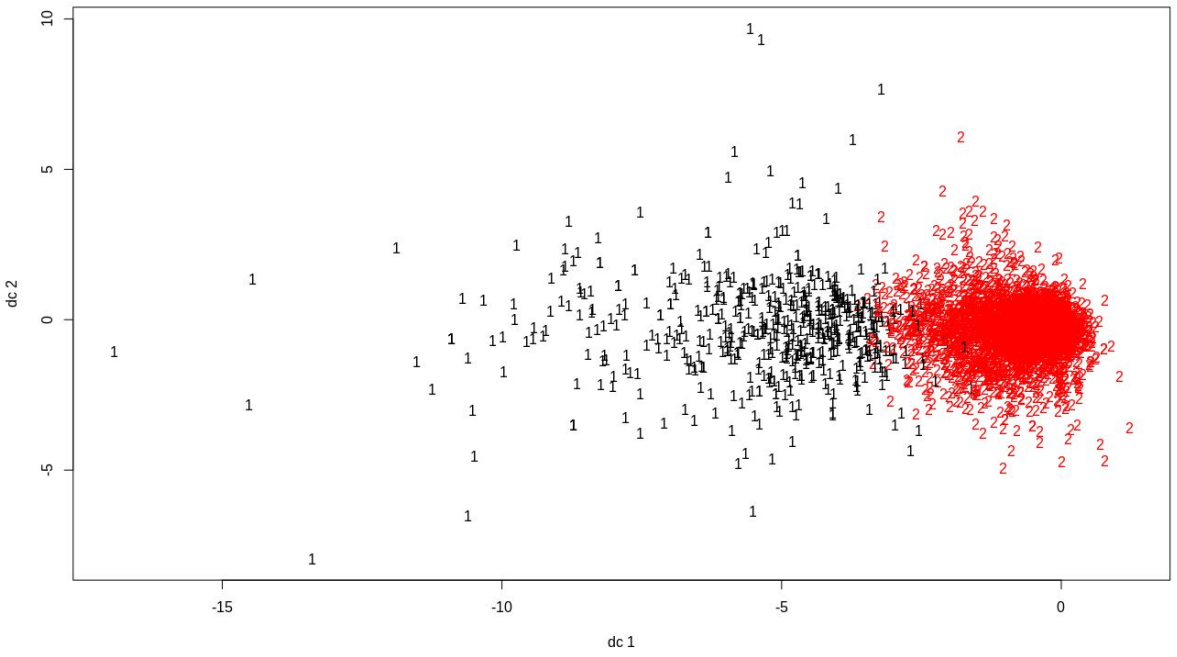
```
$wb.ratio
[1] 0.73043

$ch
[1] 87.39191

$cwidegap
[1] 15.68746 33.74080 35.23108

$widestgap
[1] 35.23108

$sindex
[1] 4.884882

$corrected.rand
[1] 0.0388109

$vi
[1] 1.702337
```

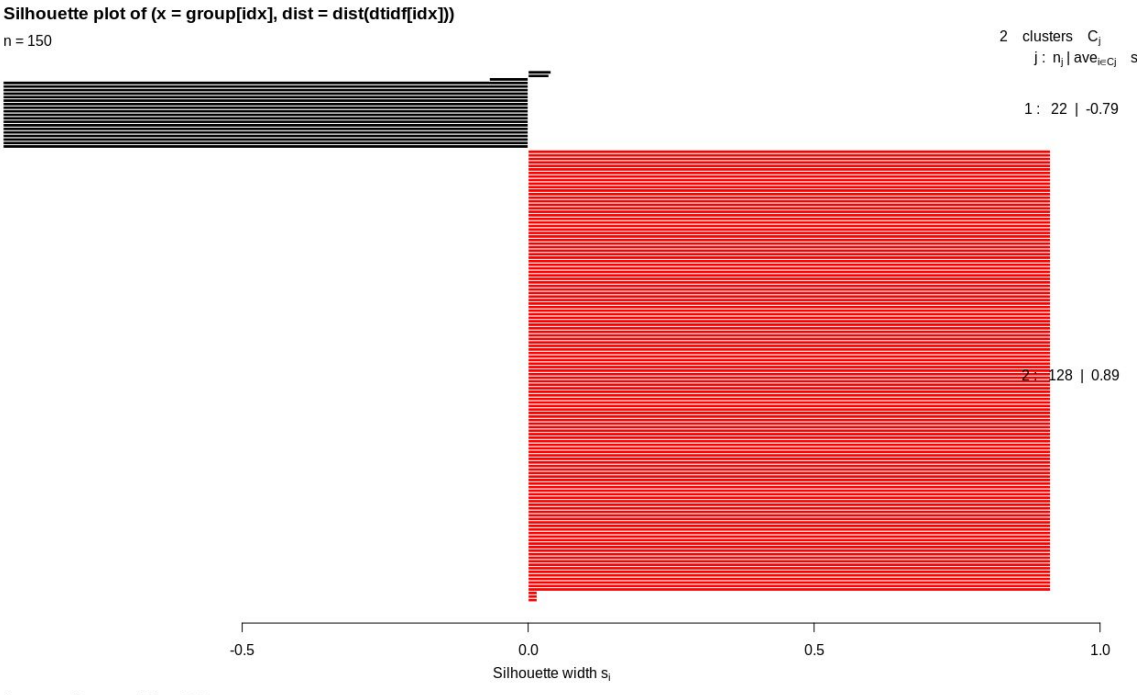
Vamos a intentar ver cuál sería el número óptimo de cluster para las k-medias por medio de la función “fviz_nbclust” que automatiza el proceso empleando como medida de varianza la suma de residuos cuadráticos, hemos querido hacer esta prueba ya que no vemos buenos resultados con 3 clusters que serían ideales para nuestro estudio ya que usamos sentimientos positivos, negativos y neutros.

fviz_nbclust (dtidf, kmeans, method = "silhouette")



Como nos indica que lo ideal serían dos clusters repetimos las pruebas con los siguientes resultados:

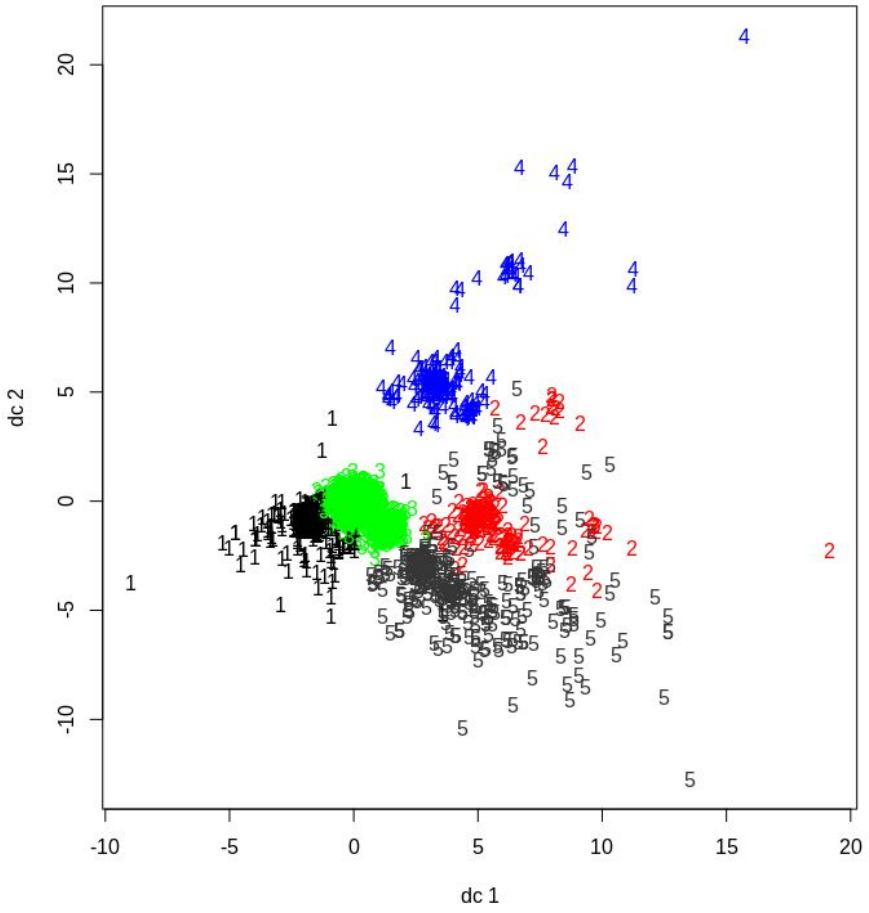




K-medoides

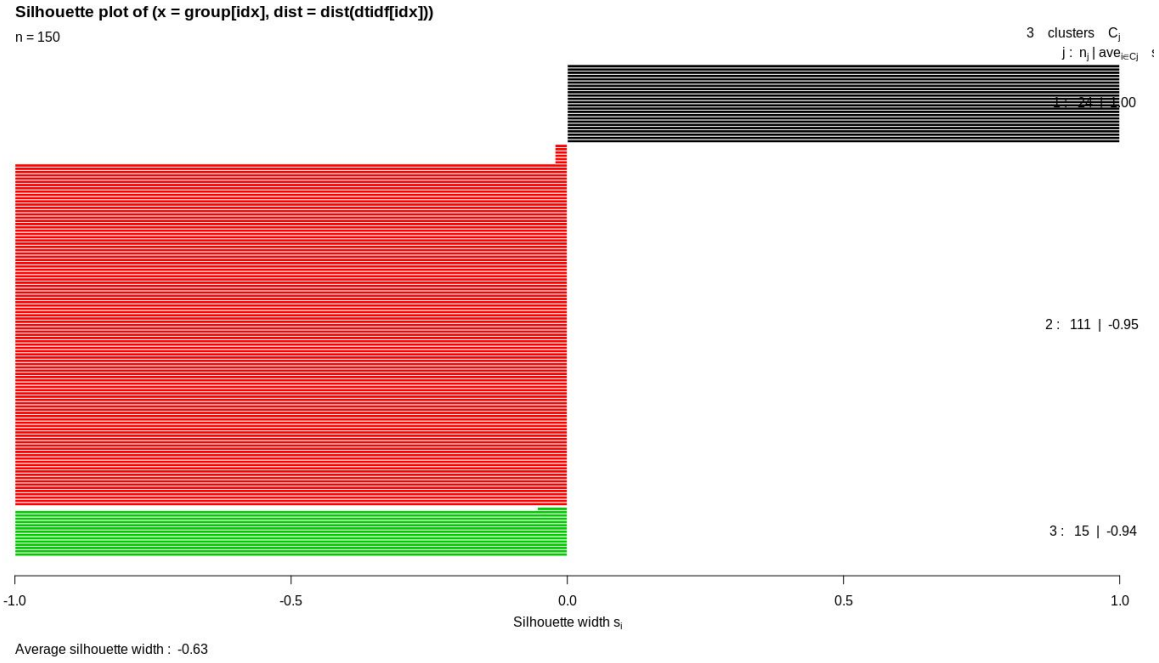
K-medoides es un método de particionado de grupos que divide los datos conformados por n objetos en k grupos, es más robusto ante el ruido y a partes aisladas ya que minimiza una suma de disimilaridades en vez de una suma de distancias euclidianas cuadradas. Vemos los resultados del mismo estudio que en los anteriores métodos

```
> pam.result=pam(dist(dtidf),5)
> idx=sample(1:dim(dtidf)[1],150)
> grupo=pam.result$clustering
> plotcluster(dtidf,grupo)
```




```
> shi= silhouette(group[idx],dist(dtidf[idx]))
```

```
> plot(shi,col=1:3)
```




```
> cluster.stats(dist(dtidf[idx]),grupo[idx])
$n
[1] 150

$cluster.number
[1] 5

$cluster.size
[1]  15  11 100  11  13

$min.cluster.size
[1] 11

$noisen
[1] 0

$diameter
[1] 0.000000 0.000000 3.427558 0.000000 3.427558

$average.distance
[1] 0.0000000 0.0000000 0.1357174 0.0000000 0.5273166

$median.distance
[1] 0 0 0 0 0

$separation
[1] 0 0 0 0 0

$average.toother
[1] 0.07616795 0.07397607 0.13436026 0.07397607 0.30599757
```

```
$separation.matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0

$ave.between.matrix
           [,1]       [,2]       [,3]       [,4]      [,5]
[1,] 0.00000000 0.00000000 0.06855115 0.00000000 0.2636583
[2,] 0.00000000 0.00000000 0.06855115 0.00000000 0.2636583
[3,] 0.06855115 0.06855115 0.00000000 0.06855115 0.3216631
[4,] 0.00000000 0.00000000 0.06855115 0.00000000 0.2636583
[5,] 0.26365828 0.26365828 0.32166311 0.26365828 0.0000000

$average.between
[1] 0.1346293

$average.within
[1] 0.1361791

$n.between
[1] 5932

$n.within
[1] 5243

$max.diameter
[1] 3.427558

$min.separation
[1] 0

$within.cluster.ss
[1] 33.87083

$clus.avg.silwidths
         1          2          3          4          5
 0.0000000  0.0000000 -0.9813501  0.0000000 -0.9246154

$avg.silwidth
[1] -0.7343667

$pearsongamma
[1] -0.00100852

$dunn
[1] 0
```

```
$dunn2
[1] 0

$entropy
[1] 1.09573

$wb.ratio
[1] 1.011511

$ch
[1] 0.715715

$cwidegap
[1] 0.000000 0.000000 3.427558 0.000000 3.427558

$widestgap
[1] 3.427558

$sindex
[1] 0
```

Conclusiones

Los resultados no son buenos, ya que la agrupación por similares de los tres clusters no es equitativa , en todos los métodos usados los datos tienden a agruparse en un mismo cluster. Hemos intentado cambiar el número de clústeres pero el resultado es similar, en conclusión el agrupamiento (aprendizaje) no supervisado para textos no es buena, creemos que deberíamos de generar un diccionario de palabras positivas, negativas y neutras para que el cluster lo tenga como referencia y así quizás obtendremos mejores resultados.

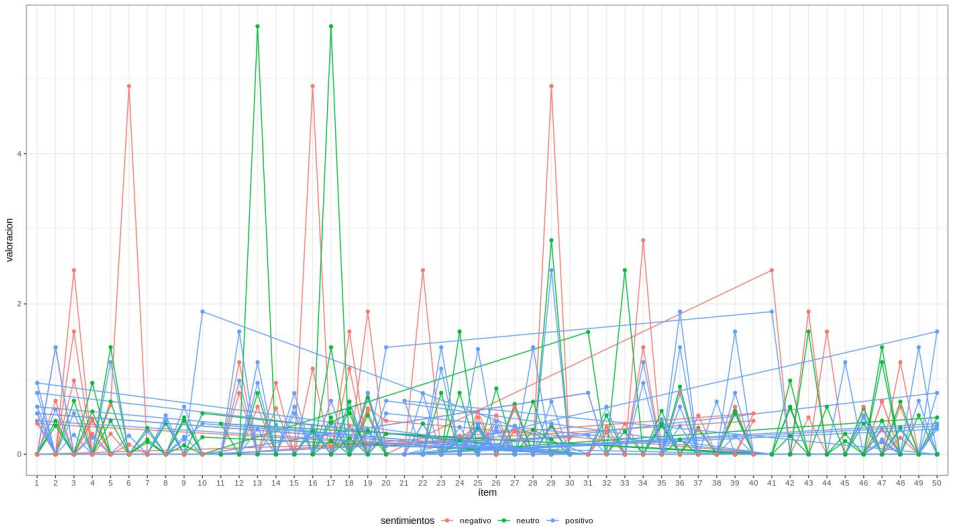
Como última prueba, para entender mejor porque no obtenemos una agrupación acorde, hemos realizado la distancia del coseno de dos documentos iguales de nuestra matriz con los mismos términos, el resultado ha sido una distancia de 0.018, cuanto mayor es el valor más se asemejan los documentos. Con esto nos afianzamos de los malos resultados.

```
  # Distancia coseno
tfidf_DT <- suppressWarnings(weightTfIdf(dtm))
terms_DT <- tfidf_DT$dimnames$Terms
id1 <- 1
id2 <- 2
doc1 <- as.vector(tfidf_DT[ , id1])
names(doc1) <- terms_DT
doc2 <- as.vector(tfidf_DT[ , id2])
names(doc2) <- terms_DT

distancia <- function(x, y){
  resultado <- x%*%y / (sqrt(x %*% x) * sqrt(y %*%y ))
```

```
  return(as.numeric(resultado))
}
distancia(doc1,doc2)
[1] 0.01800277
```

Otra forma de comprobar la distancia entre documentos es con la correlación, también nos refiere que el estudio de los documentos de texto no nos va a aportar gran información del tipo de sentimiento a partir de la opinión del usuario.

```
datos <- data.frame(sentimientos = rep(c("positivo", "negativo", "neutro"),
each = 10),
                    valoracion = c(doc1, doc2),
                    item = 1:50)
 # correlación
ggplot(data = datos, aes(x = as.factor(item), y = valoracion, colour =
sentimientos)) +
  geom_path(aes(group = sentimientos)) +
  geom_point() +
  labs(x = "ítem") +
  theme_bw() +
  theme(legend.position = "bottom")
```



Clasificación

Es una técnica de aprendizaje supervisado que permite realizar predicciones futuras basadas en comportamientos o características analizadas en datos etiquetados (salida mostrada del conjunto de datos ya conocido). Tenemos las etiquetas que las usaremos en parte del estudio que expondremos a continuación. Por otro lado pretendemos clasificar los documentos con el objetivo de descubrir el sentimiento subyacente en una colección de documentos.

Árboles de decisión

En esta técnica se realizan particiones binarias de los datos de forma recursiva, asociada a cada una de las particiones se define un criterio que determina por qué rama seguir hasta llegar a las hojas (contienen las decisiones finales). Como los posibles árboles para un conjunto de datos pueden llegar a ser muchos y no se podrían analizar todos, se utilizan criterios que en cada paso buscan la mejor opción (criterios avariciosos). Nosotros trataremos de encontrar una partición binaria que contuviese tres ramas, una para cada uno de los sentimientos.

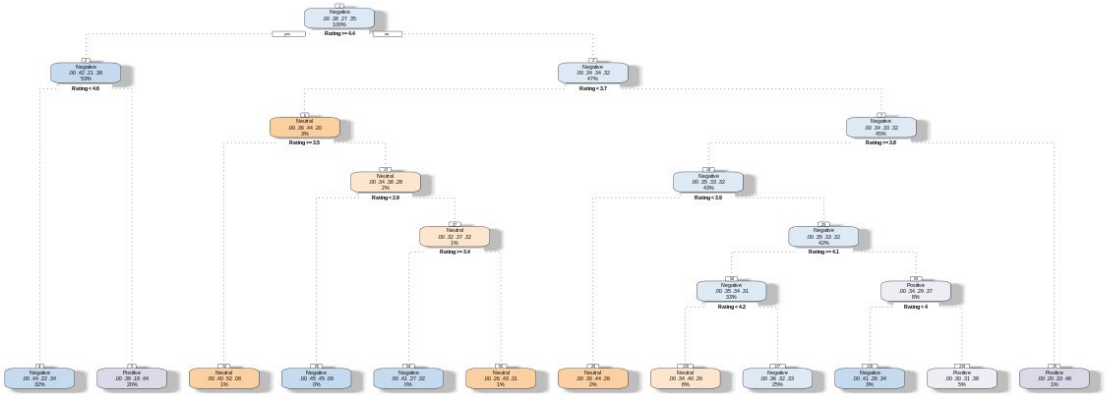
Ya vimos en el análisis que no había variables que tuvieran correlación directa con el sentimiento del usuario, aun así vamos a generar árboles para Rating, Install y Category.

Para la construcción del árbol vamos a usar la función “rpart”. El principal parámetro de esta función es la complejidad, cp. Éste permite simplificar los modelos ajustados mediante la poda de las divisiones que no merecen la pena. Otros parámetros importantes son minsplit, número mínimo de observaciones que debe haber en un nodo para que se intente una partición, y minbucket, número mínimo de observaciones de un nodo terminal. Por defecto minsplit= 20, minbucket=round(minsplit/3) y cp= 0,01.

Para este ejemplo se han tomado minsplit= 30, minbucket= 10 y cp= 0,01 ya que se está trabajando con bastantes observaciones.

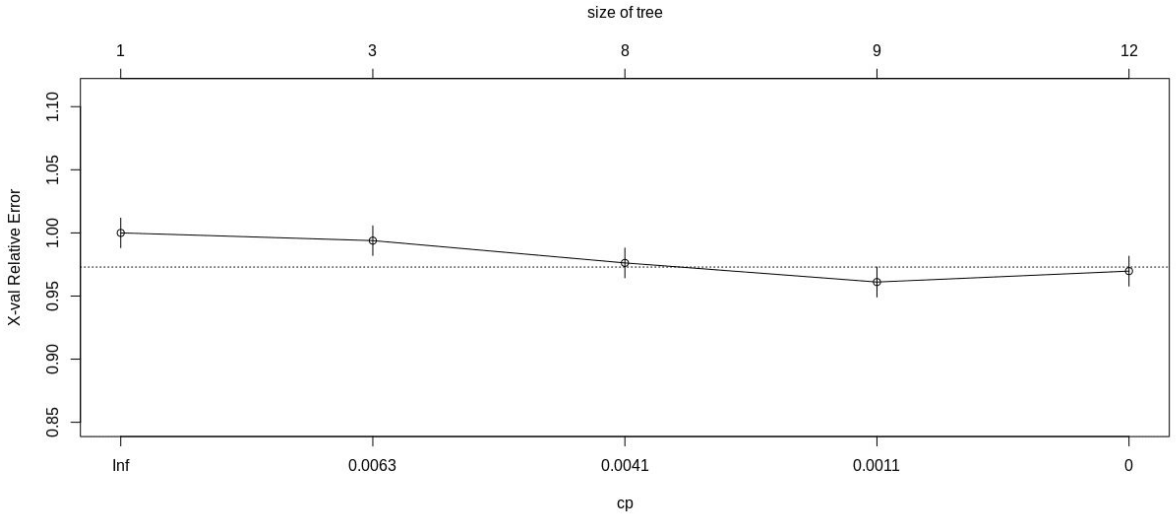
```
library("rpart")
library("rpart.plot")

c=sample(2,nrow(Trans_review),replace=TRUE,prob=c(0.7,0.3))
user_train <- Trans_review[c==1,]
user_test <- Trans_review[c==2,]

rpart <- rpart(Sentiment ~ Rating, data=user_train,
               method="class",
               parms=list(split="information"),
               control=rpart.control(minsplit=30,
                                     minbucket=10,
                                     cp=0.01,
                                     usesurrogate=0,
```

```
                                        maxsurrogate=0)
)
fancyRpartPlot(rpart)
```



Rattle 2020-ene-19 22:27:20 mati

```
plotcp(rpart)
```

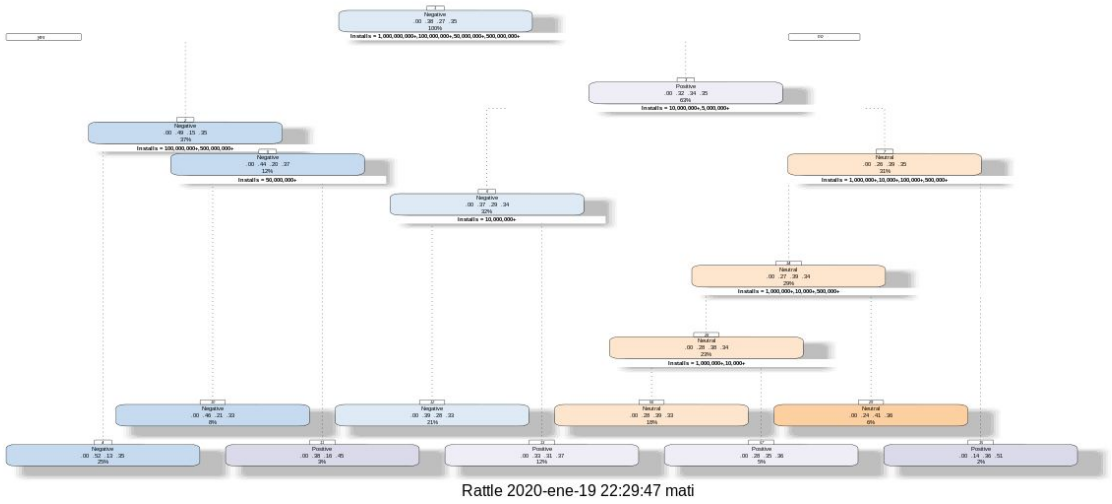



```
> printcp(rpart)

Classification tree:
rpart(formula = Sentiment ~ Rating, data = user_train, method = "class",
    parms = list(split = "information"), control = rpart.control(minsplit =
30,
        minbucket = 10, cp = 0, usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Rating

Root node error: 2777/4494 = 0.61794

n= 4494

         CP nsplit rel error  xerror     xstd
1 0.0084624      0   1.00000 1.00000 0.011730
```

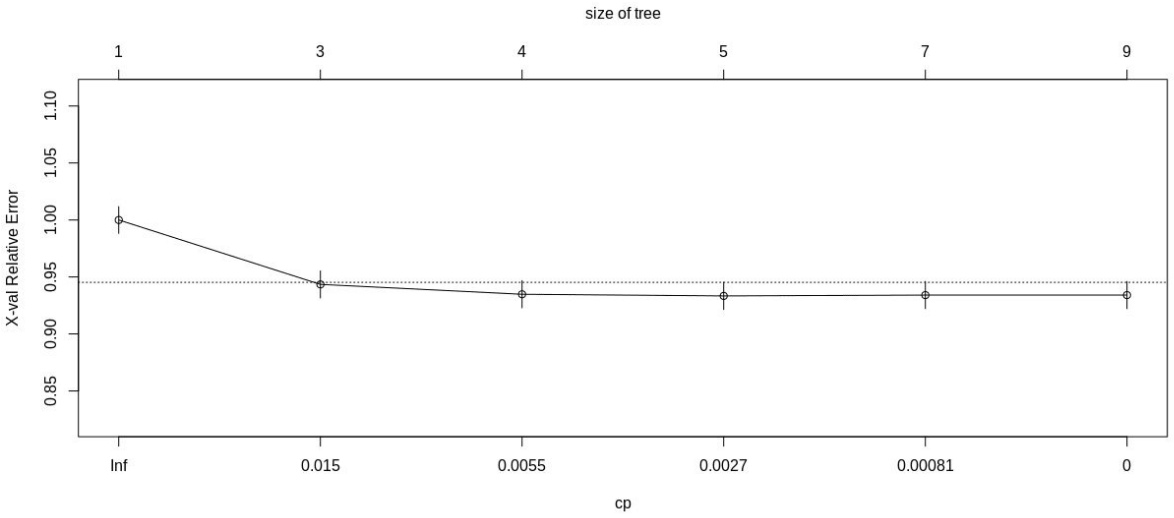
```
2 0.0046813      2   0.98308 0.99388 0.011751
3 0.0036010      7   0.95931 0.97623 0.011810
4 0.0003601      8   0.95571 0.96111 0.011855
5 0.0000000     11   0.95463 0.96975 0.011830
```

Podemos ver que el mínimo error (tendencia a 0), se alcanza en el nodo 5 (hoja).

```
rpart <- rpart(Sentiment ~ Installs, data=user_train,
               method="class",
               parms=list(split="information"),
               control=rpart.control(minsplit=30,
                                     minbucket=10,
                                     cp=0.00,
                                     usesurrogate=0,
                                     maxsurrogate=0)
)
fancyRpartPlot(rpart)
```




```
plotcp(rpart)
```

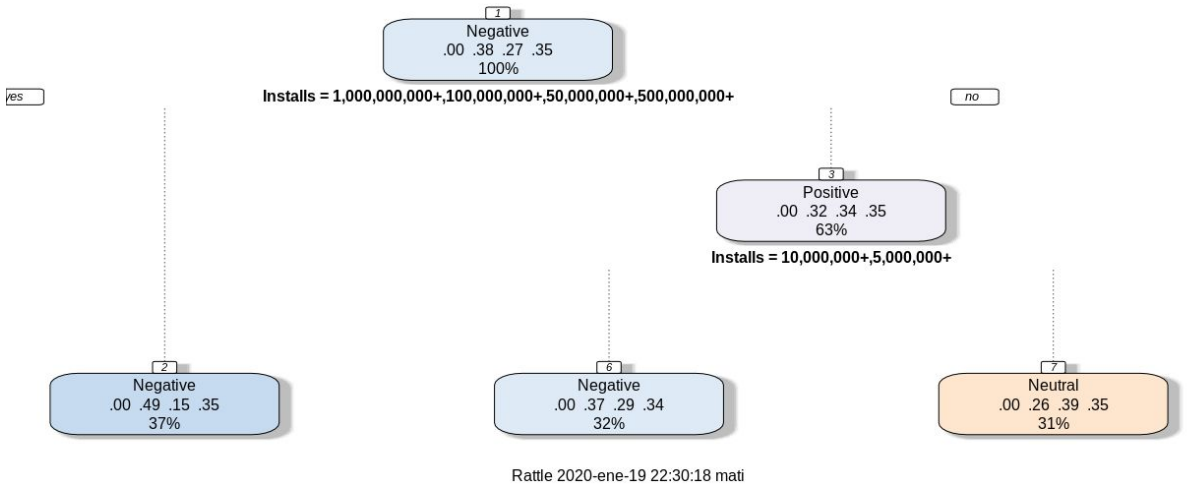



```
> printcp(rpart)
```

```
Classification tree:
rpart(formula = Sentiment ~ Installs, data = user_train, method = "class",
    parms = list(split = "information"), control = rpart.control(minsplit =
30,
        minbucket = 10, cp = 0, usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Installs

Root node error: 2777/4494 = 0.61794

n= 4494

         CP nsplit rel error  xerror     xstd
1 0.0304285      0   1.00000 1.00000 0.011730
2 0.0075621      2   0.93914 0.94346 0.011903
3 0.0039611      3   0.93158 0.93482 0.011924
4 0.0018005      4   0.92762 0.93338 0.011927
5 0.0003601      6   0.92402 0.93410 0.011925
6 0.0000000      8   0.92330 0.93410 0.011925
```

Vamos a probar a poner cp = 0.015 ya que en la imagen anterior se muestra que el error tiene cp = 0,15

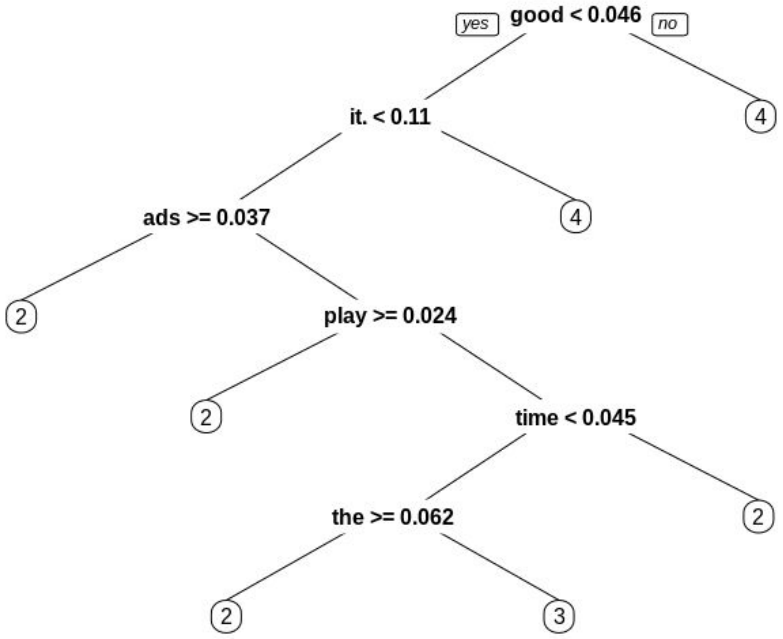
```
rpart <- rpart(Sentiment ~ Installs, data=user_train,
      method="class",
      parms=list(split="information"),
      control=rpart.control(minsplit=30,
      minbucket=10,
      cp=0.015,
      usesurrogate=0,
      maxsurrogate=0))
fancyRpartPlot(rpart)
```



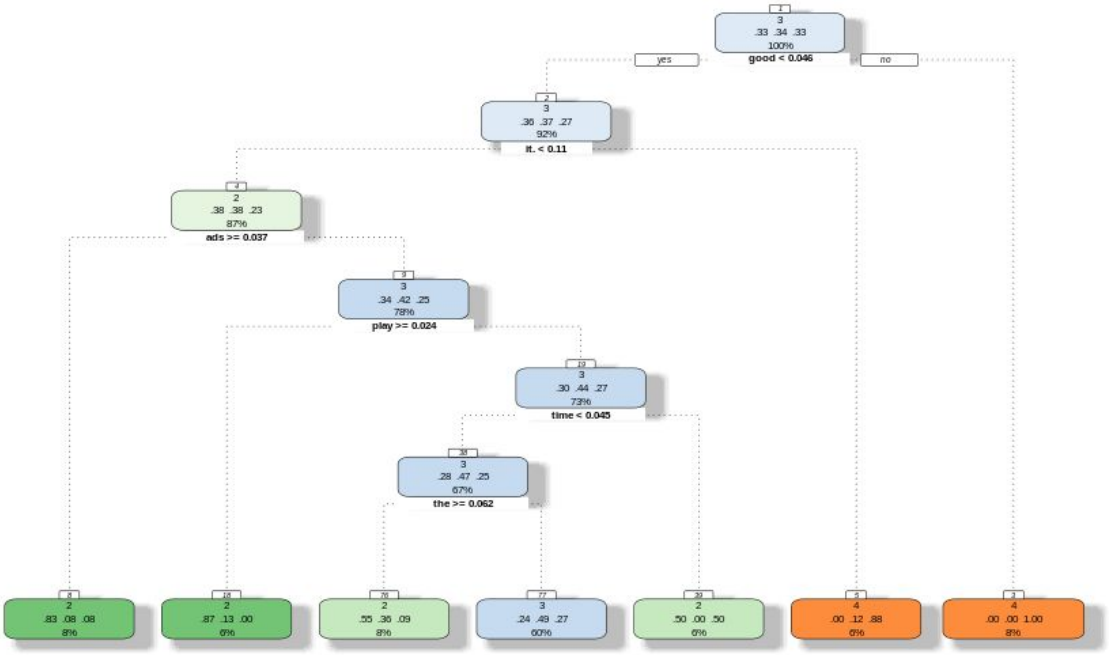
Vemos como disminuye considerablemente la dimensión del árbol.

Hacemos lo mismo con Translated_Review, nuestra columna de texto, que es nuestro real interés de estudio.

```
datos_total <- bind_rows(user_train,user_test)
corpus = Corpus(VectorSource(datos_total$Translated_Review))
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
reviews = as.data.frame(cbind(datos_total$Sentiment, as.matrix(tdm.tfidf)))
reviews <- na.omit(reviews)
preparado_train <- reviews[1:1500,]
preparado_test <- reviews[-(1:1500),]
reviews_tree = rpart(V1~., method = "class", data= preparado_train)
prp(reviews_tree)
```




```
fancyRpartPlot(reviews_tree)
```



SVM

La técnica Support Vector Machine (SVM) es una técnica originaria de clasificación aunque también se puede usar en regresión. Es uno de los mejores clasificadores dentro del aprendizaje estadístico y machine learning.

Esta técnica se basa en el concepto de hiperplano, el cual se define como un subespacio plano y afín de dimensiones p-1, por ejemplo, en un espacio tridimensional, un hiperplano es un subespacio de dos dimensiones, un plano convencional. En dimensiones p > 3 no es intuitivo visualizar un hiperplano aunque se mantiene el concepto de subespacio.

Nos hemos encontrado con muchos inconvenientes para la aplicación de esta técnica, la función create_matrix ya no es soportada y da errores, así que hemos creado directamente los contenedores generando nosotros mismos el tdidf. Las dimensiones no son coincidentes por lo que hemos tenido que ver los datos precisos y ponerlos a mano.

```
corpus = Corpus(VectorSource(user_train$Translated_Review))
tdm <- DocumentTermMatrix(corpus)
#tdm.tfidf <- suppressWarnings(weightTfIdf(tdm))

container <- create_container(tdm, t(user_train$Sentiment),
                              trainSize = 1:2996,
                              virgin = FALSE)

models <- train_models(container,algorithms=c("SVM"))
```

```
results <- classify_models(container, models)

out = data.frame(model_sentiment = results$SVM_LABEL,
                 model_prob = results$SVM_PROB,
                 actual_party = user_train$label[1:2999])
(z = as.matrix(table(out[,1], out[,3])))   # display the confusion matrix.
```

	Negative	Neutral	Positive
Negative	833	18	34
Neutral	91	957	78
Positive	40	15	933

Podemos observar que los datos son equitativos.

```
> (pct = round(((z[1,1] + z[2,2] + z[3,3])/sum(z))*100, 2))
[1] 90.73
```

La precisión (accuracy) para datos de entrenamiento es del 90.73%, proporción entre las predicciones correctas que ha realizado el modelo y el total de predicciones. Explicamos la fórmula con más detalle, hemos creado un contenedor con la matriz de documentos términos y la variable independiente sentimientos. Hemos entrenado el contenedor para obtener un modelo y generamos la matriz de confusión.

La matriz de confusión de un problema de clase n, es una matriz n*n donde las filas son las clases reales de nuestros datos y las columnas son las clases previstas por el modelo, sirve para mostrar cuando una clase es confundida con otra.

Ahora hacemos la parte de user_test

```
text = user_train$Translated_Review
cor = Corpus(VectorSource(text)) # crea otra vez el corpus
dtm <- DocumentTermMatrix(cor, list(bounds= list(global= c(5,Inf))))
dtm.test <- suppressWarnings(weightTfIdf(dtm))
row.names(dtm.test) = (nrow(dtm)+1):(nrow(dtm)+nrow(dtm.test))
dtm.f = c(tdm, dtm.test)
training_codes.f = c(training_codes,
                     rep(NA, length(user_train)))

container.f = create_container(dtm.f,
                       t(training_codes.f), trainSize=1:nrow(tdm),
                       testSize = 1:1291, virgin = T)
model.f = train_models(container.f, algorithms = c("SVM"))
predicted <- classify_models(container.f, model.f)
out = data.frame(model_sentiment = predicted$SVM_LABEL,
                 model_prob = predicted$SVM_PROB,
```

```
                text = user_test$label)
(z = as.matrix(table(out[,1], out[,3])))

   Negative Neutral Positive
  1       14     211       18
  2       32      54       25
  3      396     161      380

(pct = round(((z[1,1] + z[2,2] + z[3,3] )/sum(z))*100, 2))
[1] 34.7
```

La precisión para los datos de test es muy inferior a la de entrenamiento, suponemos que esta técnica funciona mejor con gran cantidad de datos.

KNN

En el algoritmo del vecino más cercano, se calcula la similitud entre el documento que se desea clasificar y todos los documentos que pertenecen a data_train, una vez se localiza un documento de entrenamiento similar, se le asigna la misma categoría (sentimiento).

Esté método es eficaz cuando el número de categorías es muy alto y en nuestro caso no lo es. La función knn tomará el conjunto de entrenamiento y de test sin la variable de respuesta , devolverá la predicción de clasificación del test.

Tenemos dos librerías para usar, nosotros nos decantamos por library(“class”).

```
library("class")

set.seed(3)

c=sample(2,nrow(user_reviews),replace=TRUE,prob=c(0.7,0.3))
data_train <- user_reviews[c==1,]
data_test <- user_reviews[c==2,]

colnames(data_train)<-c("text", "label")
colnames(data_test)<-c("text", "label")

# si los datos no son numericos introduce NAS por defecto
data_train$text <- as.numeric(data_train$text)
data_test$text <- as.numeric(data_test$text)
trainClass<-data_train[,"label"]
trueClass<-data_test[,"label"]
knnClass <- knn (data_train, data_test, trainClass)
# Matriz de confusión:
nnTabla <- table ("1-NN" = knnClass, Reuters = trueClass); nnTabla
    Reuters
```

```
1-NN          Negative Neutral Positive
   Negative        190     102       102
   Neutral         131     221       102
   Positive        107      97       253
sum(diag(nnTabla))/nrow(data_test)
[1] 0.5243446
```

Hemos obtenido una precisión del 50,43%, como ya habíamos comentado esta técnica es buena con un gran número de categorías y nosotros hemos usado tres categorías, teniendo esto en cuenta los resultados nos parecen aceptables.

Naive Bayes

En la técnica de Naive Bayes usamos las probabilidades condicionales de las palabras en un texto para determinar a qué categoría pertenece por medio del teorema de Bayes.

Las probabilidades condicionales de cada palabra se calcula por separado como si fueran independientes entre ellas asumiendo que esta probabilidad no afecta a las palabras que le acompañan.

Posteriormente se calcula la probabilidad conjunta de todas las palabras mediante el producto de las mismas para determinar la pertenencia a una categoría. Esto se hará para cada documento hasta clasificarlos todos.

Dividimos el dataset en datos de entrenamiento y test:

```
c=sample(2,nrow(user_reviews),replace=TRUE,prob=c(0.7,0.3))
user_train <- user_reviews[c==1,]
user_test <- user_reviews[c==2,]
```

Calculamos los corpus y la matriz de documentos términos de los datos de entrenamiento y test.

```
corpus_train <- Corpus(VectorSource(user_train))
corpus_test <- Corpus(VectorSource(user_test))
dtm_train <- DocumentTermMatrix(corpus_train, list(wordLengths= c(3,12)))
dtm_test <- DocumentTermMatrix(corpus_test, list(wordLengths= c(3,12)))
```

Obtenemos la matriz de términos en valores binarios en lugar de pesos

```
train_DT <- apply(dtm_train, MARGIN = 2, convert_binary)
test_DT <- apply(dtm_test, MARGIN = 2, convert_binary)
```

Función usada para convertir en valor binario:

```
convert_binary <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}
```

Entrenamos el clasificador con el conjunto de entrenamiento:

```
classifier <- naiveBayes(train_DT, as.factor(user_reviews$Sentiment), laplace
= 1)
```

Generamos las predicciones sobre el conjunto de entrenamiento y test

```
pred_train <- predict(classifier, newdata=train_DT)
pred_test <- predict(classifier, newdata=test_DT)
table(pred=pred_train,real=user_reviews$Sentiment)
```

```
          real
pred       Negative Neutral Positive
  Negative      728       8       14
  Neutral       309    1067      336
  Positive       21       4      716
```

```
table(pred=pred_test,real=user_reviews$Sentiment)
```

```
          real
pred       Negative Neutral Positive
  Negative      258      25       61
  Neutral       120     382      140
  Positive       64      14      233
```

Por último calculamos los errores cometidos en ambos conjuntos:

```
error_train <- mean(pred_train != user_reviews$Sentiment)
error_train
21.6047455510459  %
error_test <- mean(pred_test != user_reviews$Sentiment)
error_test
32.6908249807247  %
```

Destacamos la cantidad de tiempo invertido para la ejecución de una matriz tan grande, así que repetimos la prueba para una pequeña cantidad de palabras, las 10 más usadas en los textos, para ello introducimos:

```
freq.words <- findFreqTerms(dtm_train, 10)
```

Tendriamos que recalcular las matrices de terminos en función de las palabras más usadas:

```
dtm_freq_train <- DocumentTermMatrix(corpus_train, control=list(dictionary =
freq.words))
dtm_freq_test <- DocumentTermMatrix(corpus_test, control=list(dictionary =
freq.words))

 train_DT <- apply(dtm_train, MARGIN = 2, convert_binary)
 test_DT <- apply(dtm_test, MARGIN = 2, convert_binary)
```

Con este cambio obtenemos el siguiente resultado:

```
table(pred=pred_train,real=user_reviews$Sentiment)
          real
pred       Negative Neutral Positive
  Negative      678      45       73
  Neutral       268     981      301
  Positive       79      19      672

table(pred=pred_test,real=user_reviews$Sentiment)
          real
pred       Negative Neutral Positive
  Negative      268      36       70
  Neutral       135     410      136
  Positive       72       9      248

error_train
25.1925545571245  %
error_test
33.092485549133  %
```

La diferencia de error entre los 10 palabras más usadas y el conjunto de todas las palabras es mínimo y la diferencia de tiempo es muy grande.

Hemos dividido nuestros datos en test y entrenamiento, con los datos de entrenamiento hemos ajustado nuestro modelo, después hemos aplicado el modelo a los datos de test para analizar cuántos de ellos fueron clasificados correctamente.

Asociación

La técnica de asociación tiene como objetivo encontrar relaciones dentro de un conjunto de transacciones (documentos), más precisamente items o atributos que tienden a ocurrir de forma conjunta. Como algoritmo para identificar los atributos frecuentes y reglas de asociación se usa el algoritmo “apriori”.

Apriori tiene dos etapas, identificar todos los atributos que ocurren con una frecuencia por encima de un pivote y convertir esos atributos (item) en reglas de asociación.

Vamos a usar la librería “arules” que implementa el algoritmo apriori, este algoritmo puede imponer restricciones sobre las reglas generadas, vamos a comentar las que usamos.

Support es el soporte mínimo que debe tener un atributo para ser considerado frecuente (por defecto es 0,1), como en nuestros datos hay muchas palabras que se repiten considerablemente poco, definimos support = 0.00001.

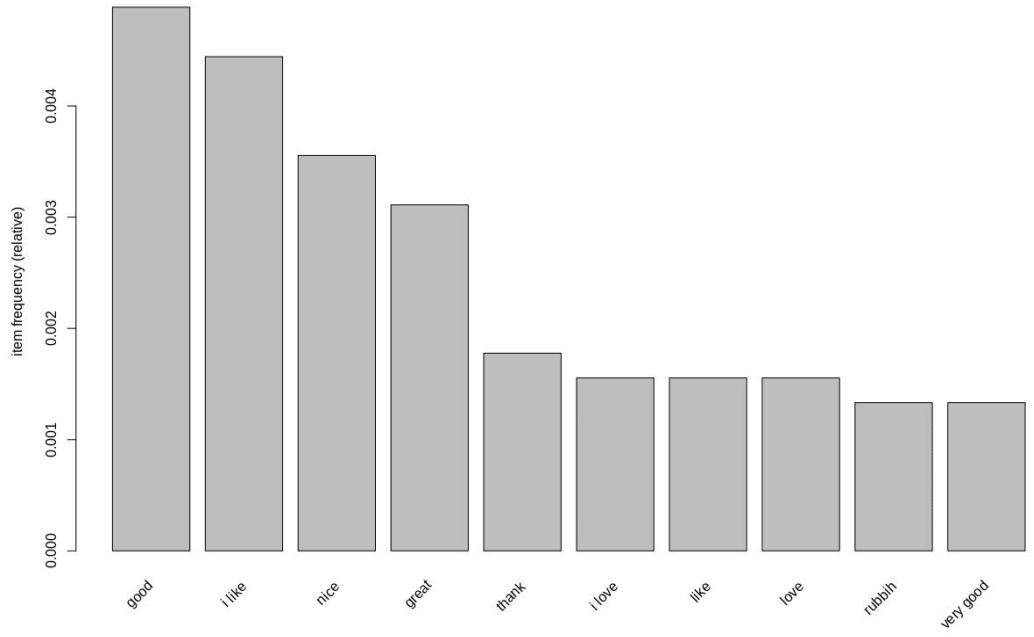
Confidence es la confianza mínima que debe de tener una regla para ser incluida en los resultados (por defecto es 0,8), hemos definido confidence = 0.2, puesto que a más confianza más dificultad y la función deja de ser efectiva para nuestros datos.

```
library(arules)
dataset = read.csv('/home/mati/Trans_review.csv', header = FALSE)
dataset = read.transactions('/home/mati/Trans_review.csv', sep = ',',
rm.duplicates = TRUE)

summary(dataset)
transactions as itemMatrix in sparse format with
 4501 rows (elements/itemsets/transactions) and
 8655 columns (items) and a density of 0.0002309006

most frequent items:
   good  i like    nice   great   thank (Other)
     22      20      16      14       8    8915

element (itemset/transaction) length distribution:
sizes
   1    2
   7 4494

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   2.000   1.998   2.000   2.000
```

```
includes extended item information - examples:
  labels
1      1
2     10
3    100
```

Vemos las frecuencias de las ocurrencias de las palabras:

```
> itemFrequencyPlot(dataset, topN = 10)
```



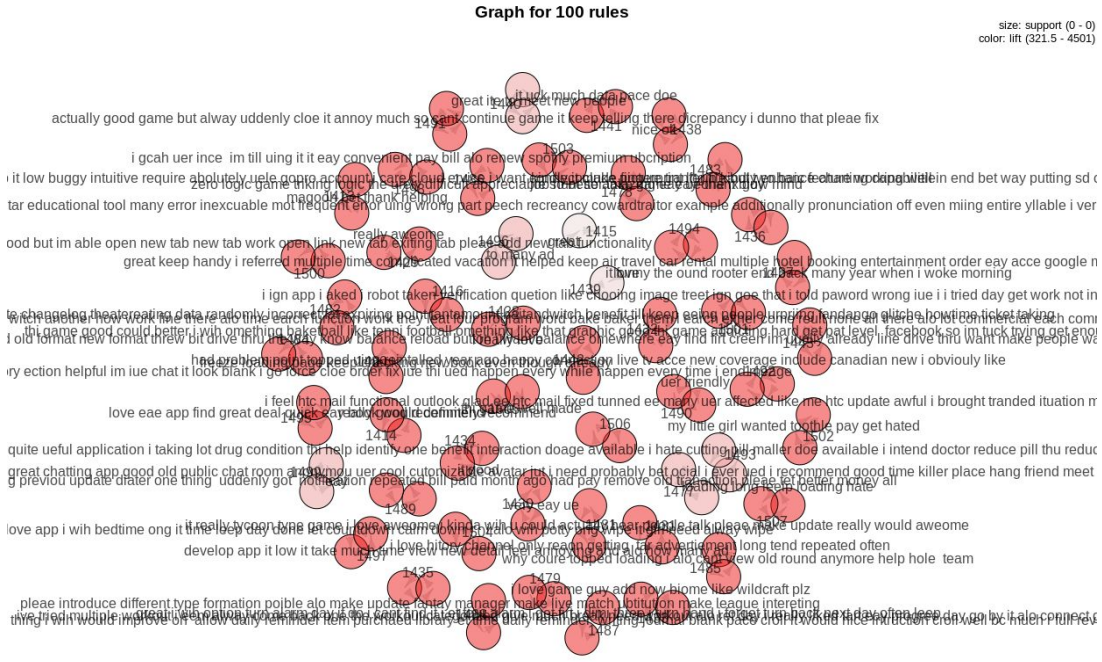

```
> rules = apriori(data = dataset, parameter = list(support = 0.00001,
confidence = 0.2))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support
minlen maxlen target   ext
        0.2    0.1    1 none FALSE            TRUE       5   1e-05
1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 0
```

```
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[8655 item(s), 4501 transaction(s)] done [0.01s].
sorting and recoding items ... [8655 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.22s].
writing ... [8875 rule(s)] done [0.13s].
creating S4 object  ... done [0.00s].


> inspect(sort(rules, by = 'lift')[1:10])
     lhs
rhs
support confidence lift count
[1]  {2958}
=> {i like except ad make impoible get much accomplihed i unintall
twice wih hopping app ad freeze photo i took that wih ad alway cauing
problem i wear}
0.0002221729          1 4501     1
[2]  {i like except ad make impoible get much accomplihed i unintall
twice wih hopping app ad freeze photo i took that wih ad alway cauing
problem i wear}
=> {2958}
0.0002221729          1 4501     1
[3]  {2959}
=> {really great app annoying ring old item till diplayed there much
choice hard actually pick thing}
0.0002221729          1 4501     1
[4]  {really great app annoying ring old item till diplayed there
much choice hard actually pick thing}
=> {2959}
0.0002221729          1 4501     1
[5]  {2960}
=> {update  till work matter i do complete wate  hour life nothing
appear work new droid turbo im card work old droid turbo unintalled
reintalled numerou time till nothing went verizon tore  aid cloud
cant help big fail} 0.0002221729          1 4501     1
[6]  {update  till work matter i do complete wate  hour life nothing
appear work new droid turbo im card work old droid turbo unintalled
reintalled numerou time till nothing went verizon tore  aid cloud
cant help big fail} => {2960}
0.0002221729          1 4501     1
[7]  {2961}
=> {i like apartment n tuff minu player many new player rude day dev
monitor new player pleae im tarting hate game bcz im ick bullie mean
```

```
people}
0.0002221729          1 4501     1
[8]  {i like apartment n tuff minu player many new player rude day
dev monitor new player pleae im tarting hate game bcz im ick bullie
mean people}
=> {2961}
0.0002221729          1 4501     1
[9]  {2962}
=> {the game kept freezing died every ingle time i hate game}
0.0002221729          1 4501     1
[10] {the game kept freezing died every ingle time i hate game}
=> {2962}
0.0002221729          1 4501     1
```

Vemos el mapa de reglas de asociación sobre los comentarios de los usuarios

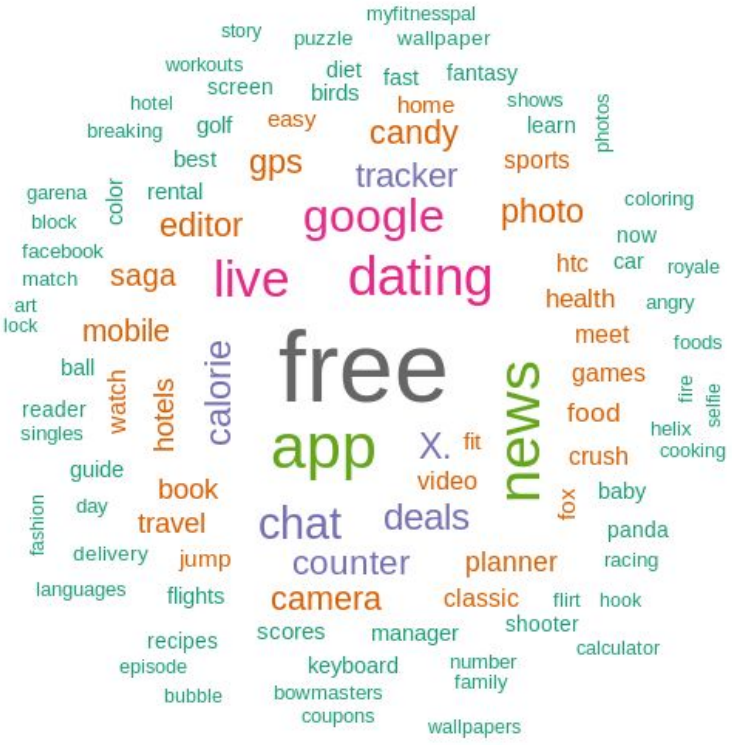
```
> plot(rules, method="graph")
```



Como podemos ver en el mapa de reglas de asociación sobre los comentarios de opinión de los usuarios, no hay agrupaciones significativas, se hacen pequeñas agrupaciones similares, ninguna destaca del resto, creemos que esto se debe a que los textos son cortos, y es difícil concretar una asociación con este tipo de textos.

Textos

Usando el codigo “tagclud.R” con alguna modificación, podemos ver las palabras más usadas en nuestro dataset, podemos ver “free” puesto que hay muchas app que son gratis.



Buscamos el mismo estudio para la columna de nuestro dataset “Translated_Review”, son los comentarios que se usan para estudiar qué tipo de sentimiento nos genera la app,

Conclusiones

Podemos ver las más grandes: game, like, great, love, good. Esto nos indica que las palabras positivas se repiten más que negativas en nuestro dataset.

Referencias Bibliográficas

https://rpubs.com/

https://www.rdocumentation.org/

http://eio.usc.es/

https://www.kaggle.com/