3 Similitud: Locality sensitive hashing

En esta parte continuaremos con la búsqueda de pares similares para colecciones de textos, y después mostraremos cómo aplicar estas técnicas para otras medidas de distancia (como distancia euclideana y coseno).

Como vimos en la parte anterior, la técnica de LSH (locality sensitive hashing) consiste en poner en cubetas a elementos que tengan hashes similares. Si diseñamos correctamente el método, entonces no es necesario hacer todas las comparaciones entre los pares, y basta examinar los elementos que compartan cubeta con otros elementos (eliminando la mayor parte de las cubetas que tendrán solo un elemento).

3.1 Análisis de la técnica de bandas

En la sección anterior dimos la primera idea como usar la *técnica de bandas* con minhashes para encontrar documentos de similitud alta, con distintos umbrales de similitud alta. Aquí describimos un análisis más detallado de la técncia

> Supongamos que tenemos un total de k minhashes, que dividimos en b bandas de tamaño r, de modo que $k = br$.
>
> - Decimos que un par de documentos *coinciden* en una banda de r hashes si coinciden en todos los hashes de esa banda.
> - Un par de documentos es un **par candidato** si por al menos coinciden en una banda (es decir, en al menos dentro de una banda todos los hashes coinciden).

Ahora vamos a calcular la probabilidad de que un par de documentos con similitud s sean un par candidato:

1. La probabilidad de que estos dos documentos coincidan en un hash particular es s, la similitud de Jaccard.

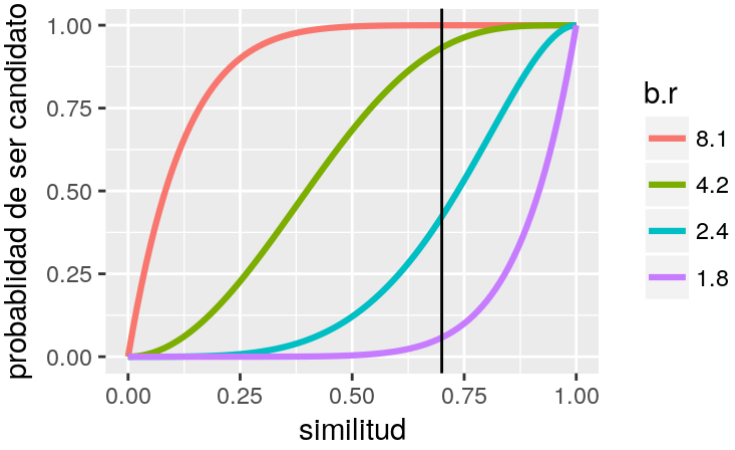
2. La probabiliad de que todos los hashes de una banda coincidan es s^r, pues seleccionamos los hashes independientemente.
3. Así que la probabilidad de que los documentos no coincidan en una banda particular es: es $1-s^r$
4. Esto implica que la probabilidad de que los documentos no coincidan en ninguna banda es $(1-s^r)^b$.
5. Finalmente, la probabilidad de que estos dos documentos sean un par candidato es $1-(1-s^r)^b$, que es la probabilidad de que coincidan en al menos una banda.

> Si la similitud de jaccard de dos documentos es s, la probabilidad de que sean un par candidato es igual a
>
> $$1-(1-s^r)^b$$
>
> .

Ejemplo

Supongamos que tenemos 8 minhashes, y que nos interesa encontrar documentos con similitud mayor a 0.7. Tenemos las siguientes posiblidades:

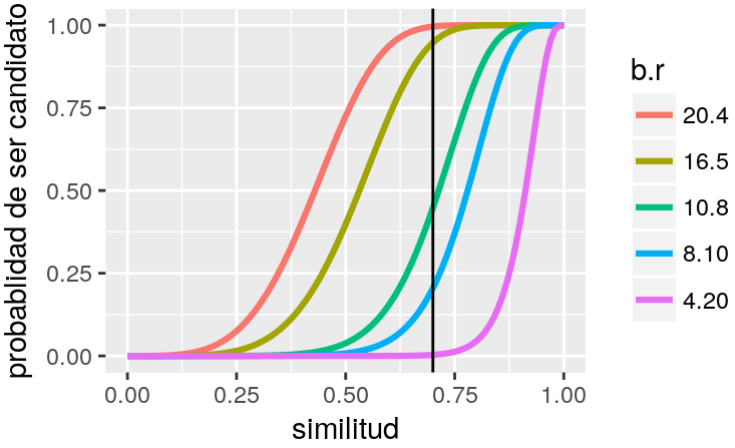
```
r <- c(1,2,4,8)
df_br <- data_frame(r = r, b = rev(r))
graficar_curvas(df_br) +
                  geom_vline(xintercept = 0.7)
```



- Con la configuración $b = 1, r = 8$ (un solo grupo de 8 hashes) es posible que no capturemos muchos pares de la similitud que nos interesa.
- Con $b = 8, r = 1$ (al menos un hash de los 8), dejamos pasar demasiados falsos positivos, que después vamos a tener que filtrar.
- Los otros dos casos son mejores para nuestro propósito. $b = 4$ produce falsos negativos que hay que filtrar, y para $b = 2$ hay una probabilidad de alrededor de 50% de que no capturemos pares con similitud cercana a 0.7

Generalmente quisiéramos obtener algo más cercano a una función escalón. Podemos acercarnos si incrementamos el número total de hashes.

```
r <- c(4, 5, 8, 10, 20)
b <- 80/r
graficar_curvas(data_frame(b, r)) +
                  geom_vline(xintercept = 0.7)
```



Observación: La curva alcanza probabilidad 1/2 cuando la similitud es

$$s = \left(1 - (0.5)^{1/b}\right)^{1/r}.$$

Y podemos usar esta fórmula para escoger valores de b y r apropiados, dependiendo de que similitud nos interesa capturar (quizá moviendo un poco hacia abajo si queremos tener menos falsos negativos).

```
lsh_half <- function(h, b){
   (1 - (0.5) ^ ( 1/b))^(b/h)
}
lsh_half(20,5)
```

```
## [1] 0.5998257
```

En (Leskovec, Rajaraman, and Ullman 2014), se utiliza la aproximación (del nivel de similitud con máxima pendiente de la curva S, según la referencia):

```
textreuse::lsh_threshold
```

```
## function (h, b)
## {
##     assert_that(is.count(h), is.count(b), check_banding(h, b))
##     (1/b)^(1/(h/b))
## }
## <environment: namespace:textreuse>
```

Que está también implementada en el paquete textreuse (Mullen 2016).

```
textreuse::lsh_threshold(20,5)
```

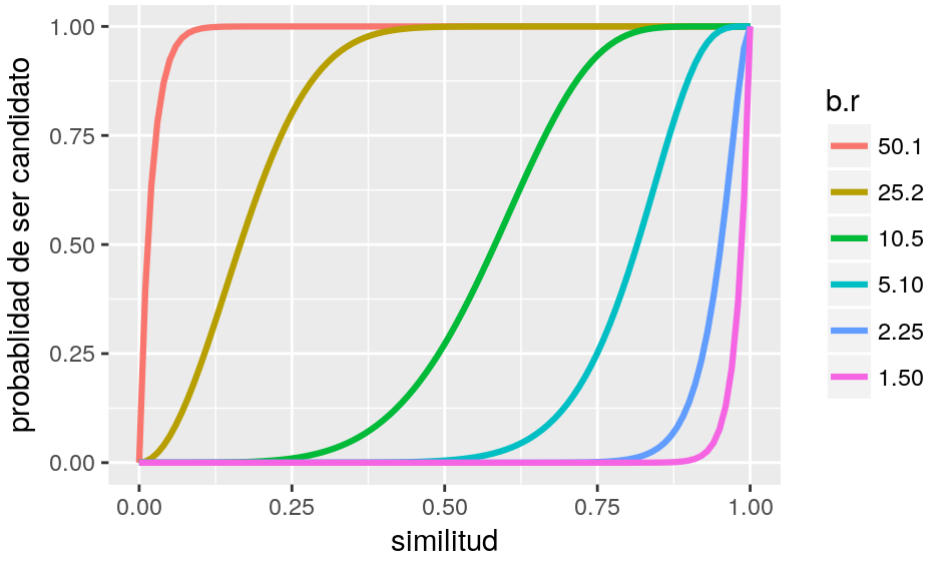
```
## [1] 0.6687403
```

Ejemplo

Supongamos que nos interesan documentos con similitud mayor a 0.5. Intentamos con 50 o 120 hashes algunas combinaciones:

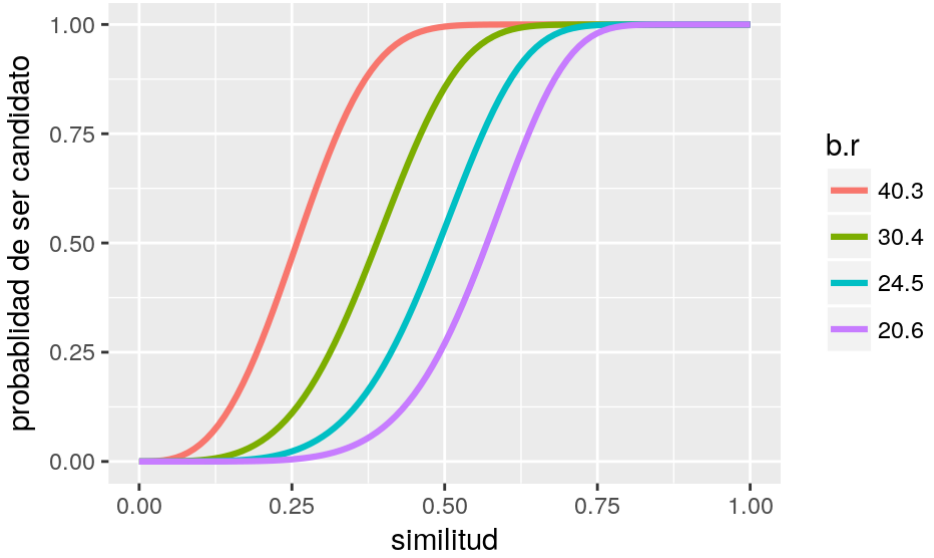
```
params_umbral <- function(num_hashes, umbral_inf, umbral_sup){
  b <- seq(1, num_hashes)
  b <- b[ num_hashes %% b == 0]
  r <- num_hashes %/% b
  combinaciones_pr <-
    data_frame(b = b, r = r) %>%
    unique() %>%
    mutate(s = (1 - (0.5)^(1/b))^(1/r)) %>%
    filter(s < umbral_sup, s > umbral_inf)
  combinaciones_pr
}

combinaciones_50 <- params_umbral(50, 0.0, 1.0)
graficar_curvas(combinaciones_50)
```



Con 120 hashes podemos obtener curvas con mayor pendiente:

```
combinaciones_120 <- params_umbral(120, 0.2, 0.6)
graficar_curvas(combinaciones_120)
```



Observación: La decisión de los valores para estos parámetros debe balancear qué tan importante es tener pares no detectados, y el cómputo necesario para calcular los hashes y filtrar los falsos positivos. La ventaja computacional de LSH proviene de hacer *trade-offs* de lo que es más importante para nuestro problema.

3.2 Resumen de LSH basado en minhashing

Resumen de (Leskovec, Rajaraman, and Ullman 2014)

1. Escogemos un número k de tamaño de tejas, y construimos el conjunto de tejas de cada documento.
2. Ordenar los pares documento-teja y agrupar por teja.
3. Escoger n, el número de minhashes. Aplicamos el algoritmo de la clase anterior (teja por teja) para calcular las firmas minhash de todos los documentos.
4. Escoger el umbral s de similitud que nos ineresa. Escogemos b y r (número de bandas y de qué tamaño), usando la fórmula de arriba hasta obtener un valor cercano al umbral. Si es importante evitar falsos negativos, escoger valores de b y r que den un umbral más bajo, si la velocidad es importante entonces escoger para un umbral más alto y evitar falsos positivos. Mayores valores de b y r pueden dar mejores resultados, pero también requieren más cómputo.
5. Construir pares similares usando LSH
6. Examinar las firmas de cada par candidato y determinar si la fracción de coincidencias sobre todos los minhashes es satisfactorio. Alternativamente (más preciso), calcular directamente la similitud de jaccard a partir de las tejas originales.

Alternativamente, podemos:

2. Agrupar las tejas de cada documento
3. Escoger n, el número de minhashes. Calcular el minhash de cada documento aplicando una función hash a las tejas del documento. Tomar el mínimo. Repetir para cada función hash.

3.3 Ejemplo: artículos de wikipedia

En este ejemplo intentamos encontrar artículos similares de wikipedia usando las categorías a las que pertenecen. En lugar de usar tejas, usaremos categorías a las que pertenecen. Dos artículos tienen similitud alta cuando los conjuntos de categorías a las que pertenecen es similar. (este el ejemplo original).

```
head -20 ../datos/similitud/wiki-100000.txt
```

```
## # 2012-06-04T11:00:11Z
## Autism Autism
## Autism Communication_disorders
## Autism Mental_and_behavioural_disorders
## Autism Neurological_disorders
## Autism Neurological_disorders_in_children
## Autism Pervasive_developmental_disorders
## Autism Psychiatric_diagnosis
## Autism Learning_disabilities
## Anarchism Anarchism
## Anarchism Political_culture
## Anarchism Political_ideologies
## Anarchism Social_theories
## Anarchism Anti-fascism
## Anarchism Greek_loanwords
## Agricultural_science Agronomy
## Albedo Climate_forcing
## Albedo Climatology
## Albedo Electromagnetic_radiation
## Albedo Radiometry
```

Primero hacemos una versión en memoria usando *textreuse*

```
library(textreuse)
limpiar <- function(lineas,...){
  df_lista <- str_split(lineas, ' ') %>%
    keep(function(x) x[1] != '#') %>%
    transpose %>%
    map(function(col) as.character(col))
  df <- data_frame(articulo = df_lista[[1]],
                   categorias = df_lista[[2]])
  df
}
filtrado <- read_lines_chunked('../datos/similitud/wiki-100000.txt',
                    skip = 1, callback = ListCallback$new(limpiar))
articulos_df <- filtrado %>% bind_rows %>%
                group_by(articulo) %>%
                summarise(categorias = list(categorias))


set.seed(99)
muestra <- articulos_df %>% sample_n(10)
muestra
```

```
## # A tibble: 10 x 2
##    articulo                    categorias
##    <chr>                       <list>
##  1 Mathematical_logic          <chr [1]>
##  2 Bock                        <chr [1]>
##  3 Paradigm_shift              <chr [6]>
##  4 William_Jardine_(merchant)  <chr [13]>
##  5 Liberation_Day_(Netherlands) <chr [5]>
##  6 V_bomber                    <chr [2]>
##  7 Ordovician                  <chr [1]>
##  8 Eureka,_Missouri            <chr [2]>
##  9 Geography_of_Jersey         <chr [2]>
## 10 Colorado                    <chr [3]>
```

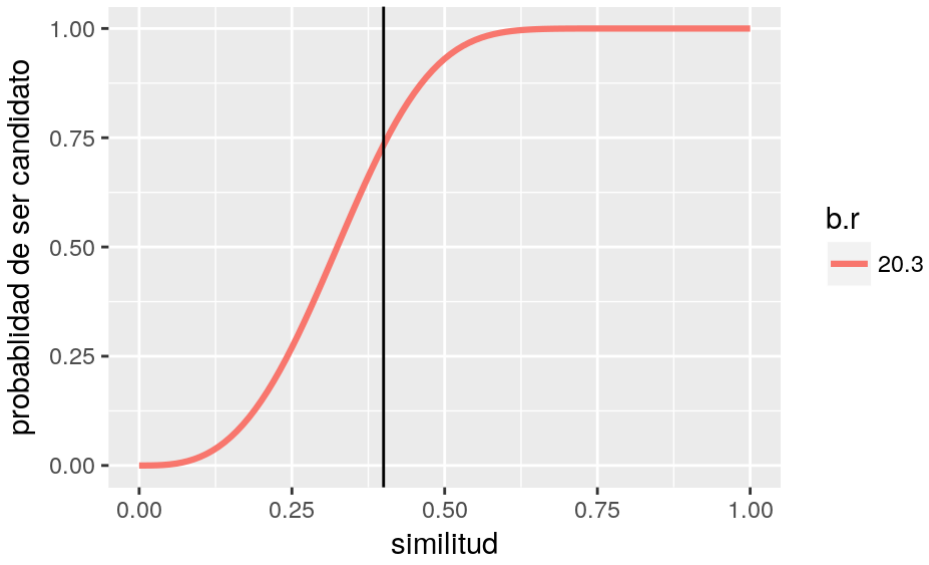
```
muestra$categorias[[10]]
```

```
## [1] "Colorado"
## [2] "States_and_territories_established_in_1876"
## [3] "States_of_the_United_States"
```

Selección de número de hashes y bandas

Ahora supongamos que buscamos artículos con similitud mínima de 0.4. Experimentando con valores del total de hashes y el número de bandas, podemos seleccionar, por ejemplo:

```
b <- 20
num_hashes <- 60
lsh_half(num_hashes, b = b)
## [1] 0.3241633
graficar_curvas(data_frame(b = b, r = num_hashes/b)) +
                geom_vline(xintercept = 0.4)
```



Tejas y cálculo de minhashes

```
options("mc.cores" = 4L)
# esta es la función que vamos a usar:
tokenize_sp <- function(x) str_split(x, ' ', simplify = TRUE)
# aunque otra opción es:
minhashes <- minhash_generator(num_hashes, seed = 1223)
# esta línea solo es necesaria porque TextReuseCorpus espera una
# línea de texto, no un vector de tokens.
textos <- articulos_df$categorias %>%
          lapply(function(x) paste(x, collapse = ' ')) %>%
          as.character
names(textos) <- articulos_df$articulo
system.time(
wiki_corpus <-  TextReuseCorpus(
                text = textos,
                tokenizer = tokenize_sp,
                minhash_func = minhashes,
                skip_short = FALSE)
)
```

```
##    user  system elapsed
##   2.720   0.170   2.941
```

```
str(wiki_corpus[[1002]])
```

```
## List of 5
##  $ content  :Class 'String'  chr "April 2002"
##  $ tokens   : NULL
##  $ hashes   : int [1:2] 1622861700 1738740796
##  $ minhashes: int [1:60] 1102660230 -1060594592 -1279335900 -1449053993 -800949525 82994
##  $ meta     :List of 4
##   ..$ hash_func   : chr "hash_string"
##   ..$ id          : chr "April_2002"
##   ..$ minhash_func: chr "minhashes"
##   ..$ tokenizer   : chr "tokenize_sp"
##  - attr(*, "class")= chr [1:2] "TextReuseTextDocument" "TextDocument"
```

Agrupar en cubetas

```
lsh_wiki <- lsh(wiki_corpus, bands = 20)
```

```
lsh_wiki %>% sample_n(20)
```

```
## # A tibble: 20 x 2
##    doc                           buckets
##    <chr>                         <chr>
##  1 Alberto_Giacometti            89ba7a5fb4d59137c9c440c87fa1e8b6
##  2 LXX                           f276ddccb9cea0f871eb3f0629f8bf04
##  3 Precession                    0cd6f0d7ca39b7109d3728505db3a4d8
##  4 Amalaric                      01fc6ed89b4bdbe0fa1681117b4385fd
##  5 Chiapas                       c24a123cea627b2a713504a59075dcca
##  6 Actium                        89f47589854d27d60b35ebfb527a8e61
##  7 High_German_languages         4c54b5330d3cc2678ce4e6edff5e4a32
##  8 Dyne                          e6db6b402ea6d91fdb65398c11325d47
##  9 Commelinales                  b7249b818dc2fd5ba1fd2b8089837682

## 10 Integer_factorization         9dd9e86c5df7b80b142a95862fde4cf4
## 11 Amos_Bronson_Alcott           4f0dbac19b5bc2b42095092156590e56
## 12 Carl_Friedrich_Gauss          60d9dffbd025f03004a53cda5244a6db
## 13 Jaguar                        16634674c3bd03982a25ef1d8f0081c7
## 14 Rashi                         601af6d65b5c624af5c4dc02293c4873
## 15 Survey_sampling               1aa0f71c59ae9a81a684b5e168733208
## 16 Mental_disorder               776f8cbedb515c8276bbd28aa1e46207
## 17 Joual                         f632ebf14fceb1d67e28d4afcb796fc6
## 18 Key_frame                     14910204dc754f612b0e1da9bfd8c7e1
## 19 Sejm_of_the_Republic_of_Poland da0d35880d59b04d738ed2270fcdc676
## 20 Hertz                         0be118f43d4300a903f7e1252cbaefcf
```

Observación: en la parte anterior sugerimos que podíamos normalizar los nombres de las cubetas. Esto también lo podemos hacer con funciones hash. Podemos usar por ejemplo el algoritmo *md5*, que está implementado para hacer hashes de objetos de R arbitrarios.

```
library(digest)
digest(c(-2341, 2221 , 21112))
```

```
## [1] "c7dde9266087f6729bfc5e39fd35f1a4"
```

```
digest(c('una' , 'dos'))
```

```
## [1] "aaf63aee22505947ae87a0ca3d992631"
```

```
x <- c(0,1); y <- c(2,3)
a <- lm(y~x)
digest(a)
```

```
## [1] "72cbfefb5619fee44c8dea89c3b9fffc"
```

Agrupamos por cubetas y filtramos las cubetas con más de un documento:

```
cubetas_df <- lsh_wiki %>%
              group_by(buckets) %>%
              summarise(candidatos = list(doc)) %>%
              mutate(num_docs = map_int(candidatos, length)) %>%
              filter(num_docs > 1)
```

```
cubetas_df <- cubetas_df %>% arrange(desc(num_docs))
nrow(cubetas_df)
```

```
## [1] 12031
```

```
sample_n(cubetas_df, 20)
```

```
## # A tibble: 20 x 3
##    buckets                          candidatos num_docs
##    <chr>                            <list>        <int>
##  1 7ae3be23e5a1873fbbdc7e1d3fc3a1ce <chr [2]>         2
##  2 bf346b66d13a6b690e1189f220a44b5c <chr [2]>         2
##  3 541d85ffd96df9a4001b6beeee8f7f76 <chr [2]>         2
##  4 c5b13723b4d2f57f68bded2f1d877175 <chr [2]>         2
##  5 8133b5bbccf9d231791eb4551541b9c3 <chr [2]>         2
##  6 ce6a8c370ad08a48dc68b9b0ddfe8170 <chr [2]>         2
##  7 4073ca3997bee23cf1ab0eaa3198d4c5 <chr [2]>         2
##  8 fd0a64bd9020a52fe86c5583d0f10777 <chr [2]>         2
##  9 1fa88661fef52bbdcdc3fce6e9f059d7 <chr [3]>         3

## 10 59925537e2a67a533b5a940049ad142d <chr [2]>         2
## 11 ec9d4c5fc25ae24eb34dfdca8882ff80 <chr [2]>         2
## 12 e2388ba29234b70ff1e4301e148461ab <chr [2]>         2
## 13 7308ad6961949768fc3a3c5d18df06c6 <chr [2]>         2
## 14 89a0060252944f4f142a48e36594723a <chr [2]>         2
## 15 7002620e7a13b8da112db6131544c07d <chr [2]>         2
## 16 ebb5cc32f94eae6880bf6897413ee1d4 <chr [4]>         4
## 17 44954141d4cc2af0901c083a6969ba6e <chr [2]>         2
## 18 8625e22883eefd93322b88a30cae1eb7 <chr [2]>         2
## 19 b509fe213b63ab7883f517a3cdee7de4 <chr [9]>         9
## 20 b12a7af4efe34fc4f23ac3d1facf5e34 <chr [2]>         2
```

```
cubetas_df$candidatos[[1]]
```

```
##   [1] "April_1"      "April_10"     "April_11"     "April_12"
##   [5] "April_13"     "April_14"     "April_15"     "April_16"
##   [9] "April_17"     "April_18"     "April_19"     "April_2"
##  [13] "April_20"     "April_21"     "April_22"     "April_23"
##  [17] "April_24"     "April_25"     "April_26"     "April_27"
##  [21] "April_28"     "April_29"     "April_3"      "April_30"
##  [25] "April_5"      "April_6"      "April_7"      "April_8"
##  [29] "April_9"      "December_1"   "December_10"  "December_11"
##  [33] "December_12"  "December_13"  "December_14"  "December_15"
##  [37] "December_16"  "December_17"  "December_18"  "December_19"
##  [41] "December_2"   "December_20"  "December_21"  "December_22"
##  [45] "December_24"  "December_25"  "December_26"  "December_27"

##  [49] "December_28"  "December_29"  "December_3"   "December_30"
##  [53] "December_31"  "December_4"   "December_5"   "December_6"
##  [57] "December_7"   "December_8"   "December_9"   "February_1"
##  [61] "February_10"  "February_11"  "February_12"  "February_13"
##  [65] "February_14"  "February_15"  "February_16"  "February_17"
##  [69] "February_18"  "February_19"  "February_2"   "February_20"
##  [73] "February_21"  "February_22"  "February_23"  "February_24"
##  [77] "February_25"  "February_26"  "February_27"  "February_28"
##  [81] "February_29"  "February_3"   "February_4"   "February_5"
##  [85] "February_6"   "February_7"   "February_8"   "February_9"
##  [89] "June_1"       "June_10"      "June_11"      "June_12"
##  [93] "June_13"      "June_14"      "June_15"      "June_16"
##  [97] "June_17"      "June_18"      "June_19"      "June_2"
## [101] "June_20"      "June_21"      "June_22"      "June_23"
## [105] "June_24"      "June_25"      "June_26"      "June_27"
## [109] "June_28"      "June_29"      "June_3"       "June_30"
## [113] "June_5"       "June_6"       "June_7"       "June_8"
## [117] "June_9"       "November_1"   "November_10"  "November_11"
## [121] "November_12"  "November_13"  "November_14"  "November_15"
## [125] "November_16"  "November_17"  "November_18"  "November_19"
## [129] "November_2"   "November_20"  "November_21"  "November_22"
## [133] "November_23"  "November_24"  "November_25"  "November_26"
## [137] "November_27"  "November_28"  "November_29"  "November_3"
```

```
## [141] "November_30"  "November_4"   "November_5"   "November_6"
## [145] "November_7"   "November_8"   "November_9"   "September_1"
## [149] "September_10" "September_11" "September_12" "September_13"
## [153] "September_14" "September_15" "September_16" "September_17"
## [157] "September_18" "September_19" "September_2"  "September_20"
## [161] "September_21" "September_22" "September_23" "September_24"
## [165] "September_25" "September_26" "September_27" "September_28"
## [169] "September_29" "September_3"  "September_30" "September_4"
## [173] "September_5"  "September_6"  "September_7"  "September_8"
## [177] "September_9"
```

```
lapply(cubetas_df$candidatos[[1]], function(articulo)
  wiki_corpus[[articulo]]$content) %>% head
```

```
## [[1]]
## Days_of_the_year April
##
## [[2]]
## Days_of_the_year April
##
## [[3]]
## Days_of_the_year April
##
## [[4]]
## Days_of_the_year April
##
## [[5]]
## Days_of_the_year April
##
## [[6]]
## Days_of_the_year April
```

```
cubetas_df$candidatos[[714]]
```

```
## [1] "Buckyball"               "Checkers"
## [3] "Czechia"                 "Danzig"
## [5] "District_of_Columbia"    "Doctors_Without_Borders"
## [7] "Manic_depression"
```

```
lapply(cubetas_df$candidatos[[714]], function(articulo)
  wiki_corpus[[articulo]]$content) %>% head
```

```
## [[1]]
## Printworthy_redirects Redirects_from_alternative_names
##
## [[2]]
## Printworthy_redirects Redirects_from_alternative_names
##
## [[3]]
## Printworthy_redirects Redirects_from_alternative_names
##
## [[4]]
## Printworthy_redirects Redirects_from_alternative_names
##
## [[5]]
## Redirects_from_alternative_names Printworthy_redirects
##
## [[6]]
## Printworthy_redirects Redirects_from_alternative_names
```

```
cubetas_df$candidatos[[911]]
```

```
## [1] "Allophone"             "Aspirated_consonant"   "Phone_(phonetics)"
## [4] "Phonetics"             "Place_of_articulation"
```

```
lapply(cubetas_df$candidatos[[911]], function(articulo)
  wiki_corpus[[articulo]]$content) %>% head
```

```
## [[1]]
## Phonetics Phonology
##
## [[2]]
## Phonetics
##
## [[3]]
## Phonetics Phonology
##
## [[4]]
## Phonetics
##
## [[5]]
## Phonetics
```

3.4 Consulta de pares candidatos.

Si tenemos un documento dado (nuevo o de la colección) y queremos encontrar candidatos similares, podemos hacerlo usando la estructura de LSH que acabamos de construir, sin tener que recorrer todos los documentos. Podemos hacer:

- Construimos la firma minhash del documento.
- Calculamos las cubetas donde este documento cae, y buscamos estas cubetas que creamos para la colección
- Extraemos los elementos que están en estas cubetas.

Ejemplo: Consulta de pares candidatos

Podemos buscar más fácilmente candidatos similares:

```
lsh_query(lsh_wiki, 'October_1')
```

```
## # A tibble: 175 x 2
##    a         b
##    <chr>     <chr>
##  1 October_1 August_1
##  2 October_1 August_10
##  3 October_1 August_11
##  4 October_1 August_13
##  5 October_1 August_14
##  6 October_1 August_15
##  7 October_1 August_16
##  8 October_1 August_17
##  9 October_1 August_18
## 10 October_1 August_19
## # ... with 165 more rows
```

```
lsh_query(lsh_wiki, 'Icosahedron')
```

```
## # A tibble: 117 x 2
##    a           b
##    <chr>       <chr>
##  1 Icosahedron Acoustics
##  2 Icosahedron Agate
##  3 Icosahedron Amaranth
##  4 Icosahedron Analgesic
##  5 Icosahedron Analysis
##  6 Icosahedron Anarchism
##  7 Icosahedron Anatomy
##  8 Icosahedron Anchor
##  9 Icosahedron Antibacterial
## 10 Icosahedron Apocrypha
## # ... with 107 more rows
```

Veamos por qué esta última lista se ve así. Examinamos dos ejemplos, y vemos que en efecto su similitud no es tan baja:

```
wiki_corpus[["Icosahedron"]]
```

```
## TextReuseTextDocument
## hash_func : hash_string
## id : Icosahedron
## minhash_func : minhashes
## tokenizer : tokenize_sp
## content : Deltahedra Platonic_solids Pyramids_and_bipyramids Greek_loanwords
```

```
wiki_corpus[["Disaster"]]
```

```
## TextReuseTextDocument
## hash_func : hash_string
## id : Disaster
## minhash_func : minhashes
## tokenizer : tokenize_sp
## content : Disasters Greek_loanwords
```

```
minhash_estimate <- function(a, b, corpus){
  mean(corpus[[a]]$minhashes == corpus[[b]]$minhashes)
}

lsh_query(lsh_wiki, 'Icosahedron') %>%
  rowwise %>%
  mutate(score = minhash_estimate(a, b, wiki_corpus)) %>%
  arrange(desc(score))
```

```
## # A tibble: 117 x 3
##    a           b              score
##    <chr>       <chr>          <dbl>
##  1 Icosahedron Dodecahedron   0.767
##  2 Icosahedron Octahedron     0.700
##  3 Icosahedron Tetrahedron    0.700
##  4 Icosahedron Platonic_solid 0.500
##  5 Icosahedron Cube           0.283
##  6 Icosahedron Acoustics      0.267
##  7 Icosahedron Agate          0.267
##  8 Icosahedron Amaranth       0.267
##  9 Icosahedron Analgesic      0.267
## 10 Icosahedron Analysis       0.267
## # ... with 107 more rows
```

Filtrar falsos positivos

Y también podemos preprocesar todos los candidatos y eliminar los falsos positivos:

```
wiki_candidatos <- lsh_candidates(lsh_wiki)
wiki_candidatos %>% nrow
```

```
## [1] 66699
```

Tenemos que evaluar estos resultados (antes tendríamos que haber evaluado alrededor de 127 millones de pares). Calculamos el score:

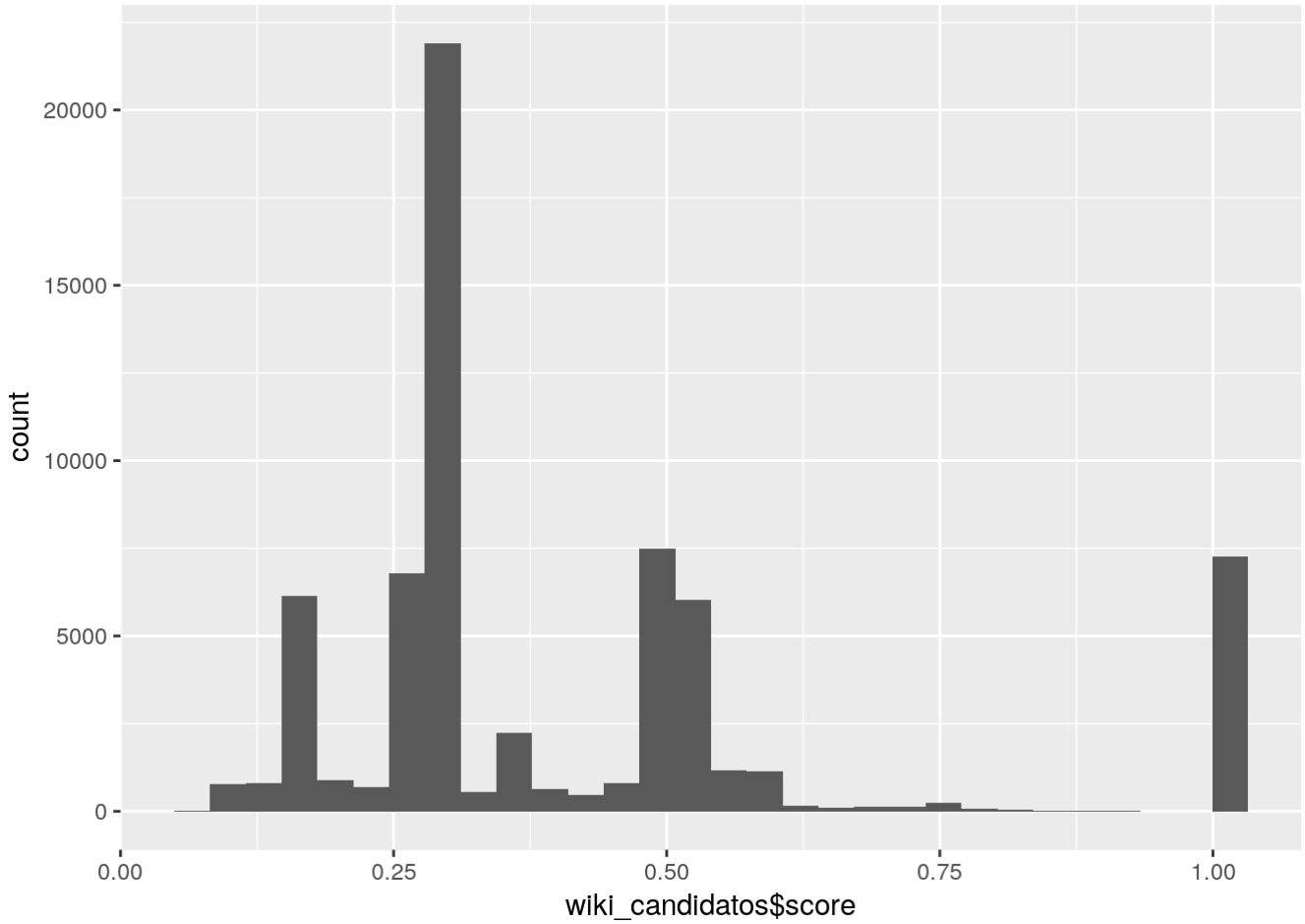
```
wiki_candidatos <-
  wiki_candidatos %>%
  rowwise %>%
  mutate(score = minhash_estimate(a, b, wiki_corpus))
wiki_candidatos %>% sample_n(20)
```

```
## # A tibble: 20 x 3
##    a                              b                                  score
##    <chr>                          <chr>                              <dbl>
##  1 Goal_line_(American_football) Play_from_scrimmage               0.500
##  2 Demographics_of_Liberia       Demographics_of_Mauritania         0.283
##  3 December_27                   May_27                             0.283
##  4 Atom                          Lemma_(mathematics)                0.167
##  5 February_12                   September_20                       0.300
##  6 Analog_television             NTSC                               0.400
##  7 Apocrypha                     Parish                             0.400
##  8 Fl%C3%A5klypa_Grand_Prix      Manufacturing_Consent:_Noam_Choms… 0.167
##  9 May_1                         October_19                         0.500
## 10 Ecdysis                       Pangenesis                         0.267
## 11 December_2                    October_15                         0.283
## 12 Paralysis                     Semantics                          0.267

## 13 December_17                   July_4                             0.283
## 14 December_29                   March_11                           0.300
## 15 November_23                   September_11                       0.533
## 16 Ume%C3%A5_University          University_of_Gothenburg           0.467
## 17 Demographics_of_Malawi        Demographics_of_Mauritius          0.267
## 18 December_24                   May_7                              0.283
## 19 Franz_Schmidt                 Richard_Wagner                     0.0667
## 20 Hormone                       Oligarchy                          0.267
```

```
qplot(wiki_candidatos$score)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
candidatos_finales <- filter(wiki_candidatos, score > 0.4)
nrow(candidatos_finales)
```

```
## [1] 25265
```

```
candidatos_finales %>%
  sample_n(200)
```

```
## # A tibble: 200 x 3
##    a                                                    b             score
##    <chr>                                                <chr>         <dbl>
##  1 April_19                                             September_4   0.533
##  2 July_4                                               May_21        0.500
##  3 Alaska                                               Pennsylvania  0.500
##  4 January_19                                           January_9     1.00
##  5 June_12                                              September_6   0.533
##  6 ASA                                                  TSR           1.00
##  7 JPEG_Network_Graphics                                Multiple-ima… 1.00
##  8 December_5                                           February_14   0.583
##  9 Aldebaran                                            Arcturus      0.567
## 10 List_of_NATO_reporting_names_for_anti-tank_missiles List_of_NATO… 1.00
## # ... with 190 more rows
```

```
lsh_query(lsh_wiki, 'Economy_of_Paraguay') %>%
  left_join(candidatos_finales) %>%
  filter(!is.na(score))
```

```
## # A tibble: 2 x 3
##   a                  b                   score
##   <chr>              <chr>               <dbl>
## 1 Economy_of_Paraguay Economy_of_Tanzania 0.467
## 2 Economy_of_Paraguay Economy_of_Thailand 0.467
```

```
lsh_query(lsh_wiki, 'Icosahedron') %>%
  left_join(candidatos_finales) %>%
  filter(!is.na(score))
```

```
## # A tibble: 3 x 3
##   a           b              score
##   <chr>       <chr>          <dbl>
## 1 Icosahedron Octahedron     0.700
## 2 Icosahedron Platonic_solid 0.500
## 3 Icosahedron Tetrahedron    0.700
```

```
lsh_query(lsh_wiki, 'Ghana') %>%
  left_join(candidatos_finales) %>%
  filter(!is.na(score))
```

```
## # A tibble: 9 x 3
##   a     b            score
##   <chr> <chr>        <dbl>
## 1 Ghana Liberia      0.483
## 2 Ghana Malawi       0.483
## 3 Ghana Mauritius    0.417
## 4 Ghana Namibia      0.800
## 5 Ghana Nigeria      0.567
## 6 Ghana Sierra_Leone 0.550
## 7 Ghana Swaziland    0.467
## 8 Ghana Tanzania     0.467
## 9 Ghana Uganda       0.433
```

3.5 Candidatos idénticos

Cuando buscamos candidatos idénticos, podemo intentar otras estrategias. Si los documentos no son muy grandes, podemos hacer hash del documento entero a un número grande de cubetas. Si el número de cubetas es suficientemente grande para la colección de texto, entonces cualquier par de documentos que caigan en una misma cubeta serán idénticos con muy alta probabilidad.

Para documentos más grandes, podemos tomar, por ejemplo, una selección al azar de posiciones en el documento y usar funciones hash. Puedes revisar otras técnicas en (Leskovec, Rajaraman, and Ullman 2014), Sección 3.9

Ejemplo

En el caso de artículos de wikipedia vimos algunas cubetas que artículos que contenían exactamente las mismas categorías. Podríamos hacer, por ejemplo:

```
articulos_df
```

```
## # A tibble: 15,976 x 2
##    articulo                              categorias
##    <chr>                                 <list>
##  1 -gry                                  <chr [1]>
##  2 ...Baby_One_More_Time                 <chr [7]>
##  3 %22Love_and_Theft%22                  <chr [6]>
##  4 %60Abdu'l-Bah%C3%A1                   <chr [12]>
##  5 %C3%81ed_mac_Cin%C3%A1eda             <chr [7]>
##  6 %C3%81lfheimr                         <chr [4]>
##  7 %C3%81satr%C3%BA_in_the_United_States <chr [1]>
##  8 %C3%86gir                             <chr [3]>
##  9 %C3%86lfheah_of_Canterbury            <chr [15]>
## 10 %C3%86lle_of_Sussex                   <chr [7]>
## # ... with 15,966 more rows
```

```
hash_categorias <- articulos_df %>%
                  mutate(hash_doc = map_chr(categorias,
                                  function(x) digest(sort(x))))
```

```
hash_categorias %>% filter(articulo %in% c('April_1','April_10'))
```

```
## # A tibble: 2 x 3
##   articulo categorias hash_doc
##   <chr>    <list>     <chr>

## 1 April_1  <chr [2]>  e7f0a0cf7a44a6da94c82bd1d06e6ab9
## 2 April_10 <chr [2]>  e7f0a0cf7a44a6da94c82bd1d06e6ab9
```

```
hash_categorias <- hash_categorias %>%
                   select(hash_doc, articulo) %>%
                   group_by(hash_doc) %>%
                   summarise(articulo = list(articulo)) %>%
                   mutate(num_docs = map_int(articulo, length)) %>%
                   filter(num_docs > 1) %>%
                   arrange(desc(num_docs))
hash_categorias
```

```
## # A tibble: 266 x 3
##    hash_doc                         articulo   num_docs
##    <chr>                            <list>        <int>
##  1 dfb11c2c1f0c529051733a628ee290e4 <chr [44]>       44
##  2 77eecf0a02ec00312ba5f64be6fe2ed3 <chr [34]>       34

##  3 536347d5b371b54ee9e037239b72e9fc <chr [30]>       30
##  4 7e4bd65708f8084f07d07871343a15ec <chr [30]>       30
##  5 3b985c050fe80210a667896c92e375dd <chr [29]>       29
##  6 854ecf9ba893037afce8831cc5255efc <chr [29]>       29
##  7 ae4ce7bed624c3b432614edcabe57292 <chr [29]>       29
##  8 e7f0a0cf7a44a6da94c82bd1d06e6ab9 <chr [29]>       29
##  9 051e445d4b9962fb7ee84df164494f14 <chr [28]>       28
## 10 07a17bdec275c6d0fccebc22b8158d7e <chr [28]>       28
## # ... with 256 more rows
```

De esta tabla podemos obtener los pares idénticos:

```
hash_categorias$articulo[[2]]
```

```
##  [1] "Abingdon"                      "Abitibi"
##  [3] "Altenberg"                     "Andersonville"
##  [5] "Antwerp_(disambiguation)"      "Bach_(disambiguation)"
##  [7] "Bourbon"                       "Buffalo"
##  [9] "Cadillac_(disambiguation)"     "Christchurch_(disambiguation)"
## [11] "Christiania"                   "Coleridge"
## [13] "Etna_(disambiguation)"         "Frank"
## [15] "Harvard_(disambiguation)"      "Isa_(disambiguation)"
## [17] "Joliet"                        "Joseph"
## [19] "Kennedy"                       "Kinderhook"
## [21] "Manhattan_(disambiguation)"    "Marathon_(disambiguation)"
## [23] "Milton"                        "Murray_Hill"
## [25] "Napoleon_(disambiguation)"     "New_England_(disambiguation)"
## [27] "Princeton"                     "Quincy"
## [29] "Riverside"                     "Roman"
## [31] "Saint_Louis"                   "Trenton"
## [33] "Utrecht_(disambiguation)"      "West_Point_(disambiguation)"
```

```
articulos_df %>% filter(articulo == 'Abingdon') %>% pull(categorias)
```

```
## [[1]]
## [1] "Place_name_disambiguation_pages"
```

Y podemos eliminar de nuestros candidatos, para reducir el trabajo de cálculo:

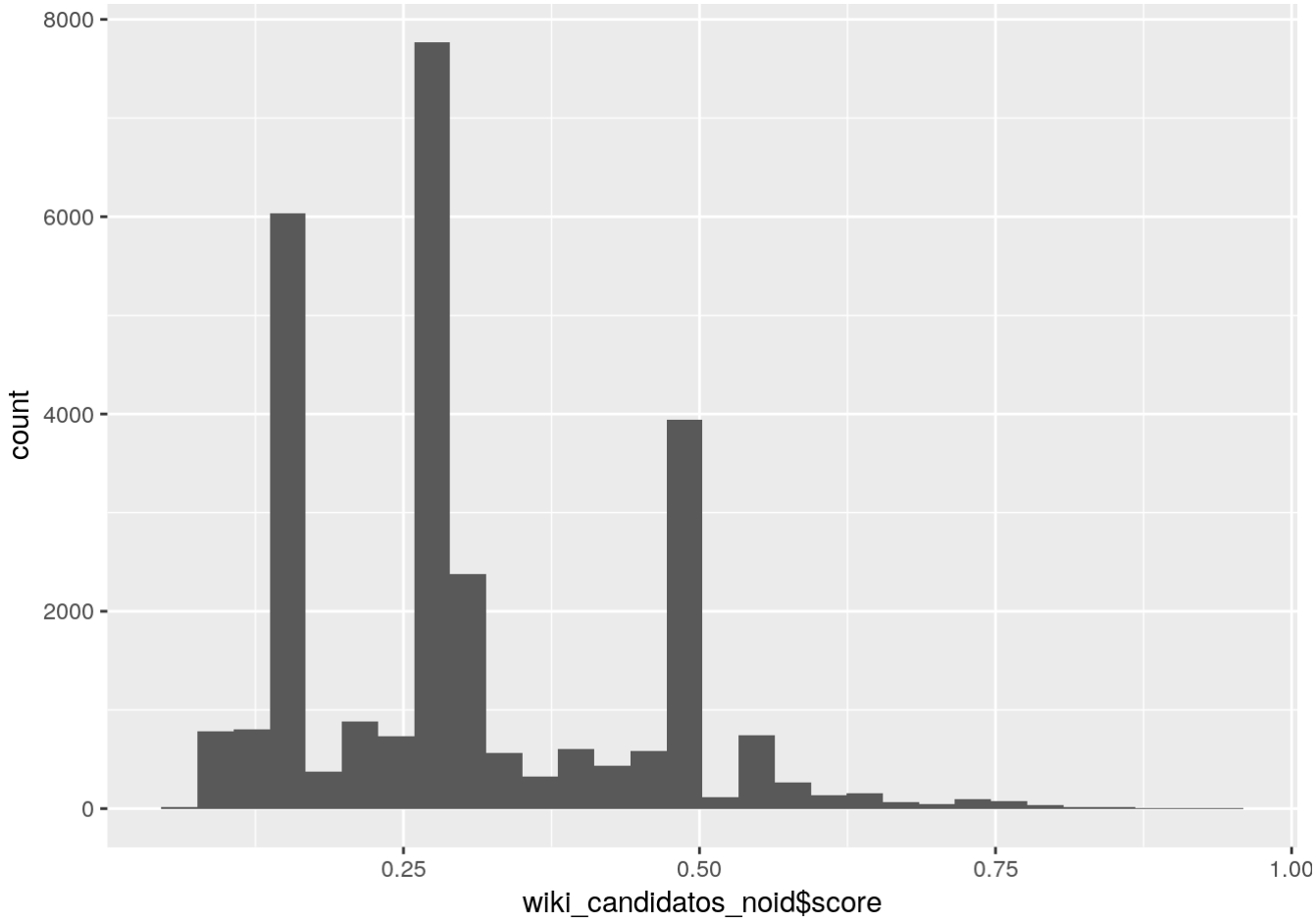
```
df_1 <- data_frame(a = unlist(hash_categorias$articulo))
wiki_candidatos_noid <- wiki_candidatos %>%
                        anti_join(df_1) %>%
                        anti_join(df_1 %>% rename(b=a))
```

```
## Joining, by = "a"
```

```
## Joining, by = "b"
```

```
qplot(wiki_candidatos_noid$score)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3.6 Medidas de distancia

La técnica de LSH puede aplicarse a otras medidas de distancia, con otras formas de hacer hash diferente del minhash. La definición de distancia puedes consultarla aquí

3.6.1 Distancia de Jaccard

Puede definirse simplemente como

$$1 - sim(a, b),$$

donde a y b son conjuntos y sim es la similitud de Jaccard.

3.6.2 Distancia euclideana

Es la distancia más común para vectores de números reales:

Si $x = (x_1, \ldots, x_p)$ y $y = (y_1, \ldots, y_p)$ son dos vectores, su norma L_2 está dada por

$$d(x, y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} = ||x - y||$$

3.6.3 Distancia coseno

La distancia coseno, definida también para vectores de números reales, no toma en cuenta la magnitud de vectores, sino solamente su dirección.

La similitud coseno se define primero como

$$sim_{cos}(x, y) = \frac{< x, y >}{||x||||y||} = \cos(\theta)$$

donde $< x, y >= \sum_{i=1}^{p} x_i y_i$ es el producto punto de x y y. Esta cantidad es igual al coseno del ángulo entre los vectores x y y (¿por qué?).

La distancia coseno es entones

$$d_{cos}(x, y) = 1 - sim_{cos}(x, y).$$

Esta distancia es útil cuando el tamaño general de los vectores no nos importa. Como veremos más adelante, una aplicación usual es comparar documentos según las frecuencias de los términos que contienen: en este caso, nos importa más la frecuencia relativa de los términos que su frecuencia absoluta (pues esta última también refleja la el tamaño de los documentos).

A veces se utiliza la distancia angular (medida con un número entre 0 y 180), que se obtiene de la distancia coseno, es decir,

$$d_a(x,y) = \theta,$$

donde θ es tal que $\cos(\theta) = d_{cos}(x,y)$.

3.6.4 Distancia de edición

Esta es una medida útil para medir distancia entre cadena. La distancia de edición entre dos cadenas $x = x_1 \cdots x_n$ y $y = y_1 \cdots y_n$ es el número mínimo de inserciones y eliminaciones (un caracter a la vez) para convertir a x en y.

Por ejemplo, la distancia entre “abcde” y “cefgh” se calcula como sigue: para pasar de la primera cadena, necesitamos agregar f, g y h (3 adiciones), eliminar d, y eliminar a,b (3 eliminaciones). La distancia entre estas dos cadenas es 6.

3.7 Teoría de funciones sensibles a la localidad

Vimos como la familia de funciones minhash puede combinarse (usando la técnica de bandas) para discriminar entre pares de baja similitud y de alta similitud.

En esta parte consideramos otras posibles familias de funciones para lograr lo mismo (hacer LSH), bajo otras medidas de distancia. Veamos las características básicas de las funciones minhash:

1. Cuando la distancia entre dos elementos x, y es baja (similitud alta), entonces $f(x) = f(y)$ tiene probabilidad alta.
2. Podemos escoger varias funciones $f_1, \ldots, f_k$ con la propiedad anterior, de manera independiente, de forma que es posible calcular la probabilidad de $f_1(x) = f_1(y)$ y $f_2(x) = f_2(y)$, y eventos combinados de este tipo.
3. Las funciones tienen que ser relativamente fáciles de calcular (comparado con calcular todos los posibles pares y sus distancias directamente).

3.8 Funciones sensibles a la localidad

Sean $d_1 < d_2$ dos valores (que interpretamos como distancias).

Una familia $\mathcal{F}$ es una familia d_1, d_2, p_1, p_2, sensible a localidad (con $p_1 > p_2$) cuando para cualquier par de elementosx, y,

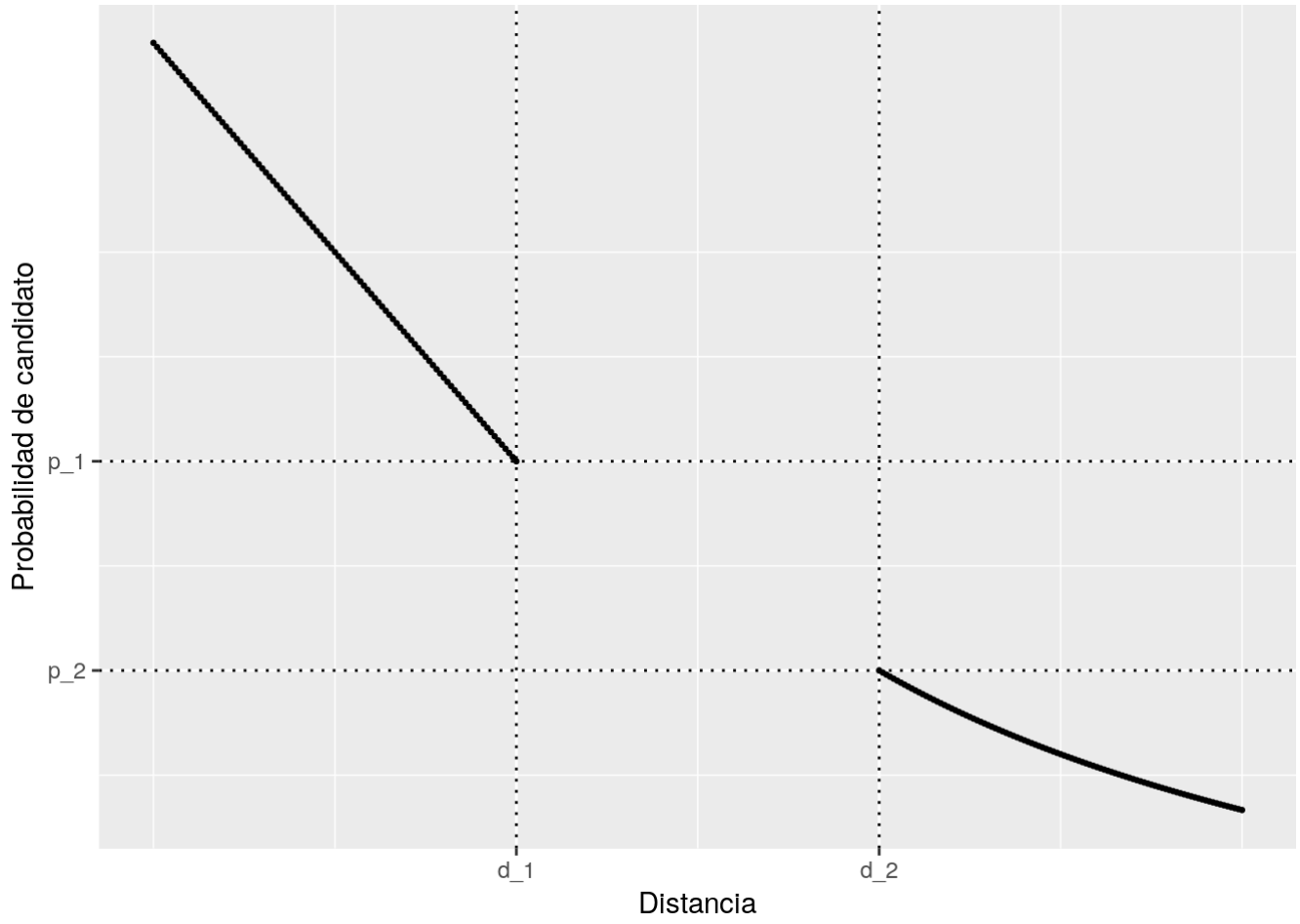
1. Si $d(x,y) \leq d_1$, entonces la probabilidad $P(f(x) = f(y)) \geq p_1$.
2. Si $d(x,y) \geq d_2$, entonces $P(f(x) = f(y)) \leq p_2$

Nótese que las probabilidades están dadas sobre la selección de f.

Estas condiciones se interpretan como sigue: cuando x y y están suficientemente cerca (d_1), la probabilidad de que sean mapeados al mismo valor por una función f de la familia es alta. Cuando x y y están lejos d_2, entonces, la probabilidad de que sean mapeados al mismo valor es baja. Podemos ver una gráfica:

```
x_1 <- seq(0, 1, 0.01)
x_2 <- seq(2, 3, 0.01)
y_1 <- -1*x_1 + 2.5
y_2 <- 2/x_2

dat_g <- data_frame(x=c(x_1,x_2),y=c(y_1,y_2))
ggplot(dat_g, aes(x=x, y=y)) + geom_point(size=0.5) +
  geom_vline(xintercept=c(1,2), linetype="dotted") +
  geom_hline(yintercept=c(1,1.5), linetype="dotted") +
  scale_x_continuous(breaks = c(1,2), labels = c('d_1','d_2')) +
  scale_y_continuous(breaks = c(1,1.5), labels = c('p_2','p_1')) +
  labs(x = 'Distancia', y ='Probabilidad de candidato')
```



3.8.1 Distancia jaccard

Supongamos que tenemos dos documentos x, y. Si ponemos por ejemplo $d_1 = 0.2$ y $d_2 = 0.5$, tenemos que si $d(x, y) = 1 - sim(x, y) \leq 0.2$, despejando tenemos $sim(x, y) \geq 0.8$, y entonces

$$P(f(x) = f(y)) = sim(x, y) \geq 0.8$$

Igualmente, si $d(x, y) = 1 - sim(x, y) \geq 0.5$, entonces

$$P(f(x) = f(y)) = sim(x, y) \leq 0.5$$

de modo que la familia de minhashes es $(0.2, 0.5, 0.8, 0.5)$ sensible a la localidad

> Para cualquier $d_1 < d_2$, la familia de funciones minhash es una familia $(d_1, d_2, 1 - d_1, 1 - d_2)$ sensible a la localidad para cualquier $d_1 \leq d_2$.

3.9 Amplificación de familias sensibles a la localidad

Con una familia sensible a la localidad es posible usar la técnica de bandas para obtener la discriminación de similitud que nos interese.

Supongamos que $\mathcal{F}$ es una familia (d_1, d_2, p_1, p_2)-sensible a la localidad. Podemos usar **conjunción** de $\mathcal{F}'$ para construir otra familia sensible a la localidad.

Sea r un número entero. Una función $f \in \mathcal{F}'$ se construye tomando $f = (f_1, f_2, \ldots, f_r)$, con f_i seleccionadas al azar de manera independiente de la familia original, de forma que $f(x) = f(y)$ si y sólo si $f_i(x) = f_i(y)$ para toda i. Esta construcción corresponde a lo que sucede dentro de una banda de la técnica de LSH.

La nueva familia $\mathcal{F}'$ es (d_1, d_2, p_1^r, p_2^r) sensible a la localidad. Nótese que las probabilidades siempre se hacen más chicas cuando incrementamos r, lo que hace más fácil discriminar pares con similitudes en niveles bajos.

Podemos también hacer **disyunción** de una familia $\mathcal{F}$. En este caso, decimos que $f(x) = f(y)$ cuando al menos algún $f_i(x) = f_i(y)$.

En este caso, la disyunción da una familia $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$ sensible a la localidad. Esta construcción es equivalente a construir varias bandas.

La idea general es ahora:

- Usando **conjunción**, podemos construir una familia donde la probabilidad p_2^r sea mucho más cercana a cero que p_1^r (en términos relativos).
- Usando **disyunción**, podemos construir una familia donde la probabilidad $1-(1-p_1^r)^b$ permanece cercana a 1, pero $1-(1-p_2^r)^b$ está cerca de cero.
- Combinando estas operaciones usando la técnica de bandas podemos construir una famlia que discrimine de manera distinta entre distancias menores a d_1 y distancias mayores a d_2.
- El costo incurrido es que tenemos que calcular más funciones para discriminar mejor.

Ejercicio

Supongamos que tenemos una familia $(0.2, 0.6, 0.8, 0.4)$ sensible a la localidad. Si combinamos con conjunción 5 de estas funciones, obtenemos una familia

$$(0.2, 0.6, 0.41, 0.025)$$

La proporción de falsos positivos es chica, pero la de falsos negativos es grande. Si tomamos 8 de estas funciones (cada una compuesta de cuatro funciones de la familia original), obtenemos una familia

$$(0.2, 0.6, 0.96, 0.08)$$

En esta nueva familia, tenemos que hacer 40 veces más trabajo para tener esta amplificación.

3.10 Distancia coseno e hiperplanos aleatorios

Construimos ahora LSH para datos numéricos, y comenzaremos con la distancia coseno. Lo primero que necesitamos es una familia sensible a la localidad para la distancia coseno.

Consideremos dos vectores, y supongamos que el ángulo entre ellos es chico. Si escogemos un hiperplano al azar, lo más probable es que queden del mismo lado del hiperplano. En el caso extremo, si los vectores apuntan exactamente en la misma dirección, entonces la probabilidad es 1.

Sin embargo, si el ángulo entre estos vectores es grande, entonces lo más probable es que queden separados por un hiperplano escogido al azar. Si los vectores son ortogonales (máxima distancia coseno posible), entonces esta probabilidad es 0.

Esto sugiere construir una familia sensible a la localidad para la distancia coseno de la siguiente forma:

- Tomamos un vector al azar v.
- Nos fijamos en la componente de la proyección de x sobre v
- Ponemos $f_v(x) = 1$ si esta componente es positiva, y $f_v(x) = -1$ si esta componente es negativa.
- Podemos poner simplemente:

$$f_v(x) = signo(< x, v >)$$

Recordatorio: La componente de la proyección de x sobre v está dada por el producto interior de x y v normalizado:

$$\frac{1}{||v||} < x, v >,$$

y su signo es el mismo de $< x, v >$.

> La familia descrita arriba (hiperplanos aleatorios) es $(d_1, d_2, (180 - d_1)/180, d_2/180)$ sensible a la localidad para la distancia angular.

Vamos a dar un argumento del cálculo: supongamos que el ángulo entre x y y es $d = \theta$, es decir, la distancia angular entre x y y es θ.

Consideramos el plano P que pasa por el origen por x y y. Si escogemos un vector al azar (cualquier dirección igualmente probable), el vector produce un hiperplano perpendicular (son los puntos z talqes que $< z, v >= 0$) que corta al plano P en dos partes. Todas las direcciones de corte son igualmente probables, así que la probabilidad de que la dirección de corte separe a x y y es igual a $2\theta/360$ (que caiga en el cono generado por x y y). Si la dirección de corte separa a x y y, entonces sus valores $f_v(x)$ y $f_v(y)$ no coinciden, y coinciden si la dirección no separa a x y y. Así que:

1. $d(x, y) = d_1 = \theta$, entonces $P(f(x) = f(y)) = d_1/180$.

Por otra lado,

2. $d(x, y) = d_2$, entonces $P(f(x) \neq f(y)) = 1 - d_2/180$.

Ejemplo: similitud coseno por fuerza bruta

Comenzamos con un ejemplo simulado.

```
set.seed(101)
mat_1 <- matrix(rnorm(300 * 1000) + 3, ncol = 1000)
mat_2 <- matrix(rnorm(600 * 1000) + 0.2, ncol = 1000)
df <- rbind(mat_1, mat_2) %>% data.frame %>%
          add_column(id_1 = 1:900, .before = 1)
head(df[,1:5])
```

```
##   id_1       X1       X2        X3       X4
## 1    1 2.673964 1.834143 2.6106873 3.057390
## 2    2 3.552462 2.760201 1.9273515 2.912274
## 3    3 2.325056 4.184383 2.6046536 2.545218
## 4    4 3.214359 3.207508 2.1550124 4.456546
## 5    5 3.310769 2.956367 2.3440595 4.618840
## 6    6 4.173966 2.028922 0.4930086 2.071095
```

Tenemos entonces 1000 variables distintas y 900 casos, y nos interesa filtrar aquellos pares de similitud alta.

Definimos nuestra función de distancia

```
norma <- function(x){
  sqrt(sum(x ^ 2))
}
dist_coseno <- function(x, y){
  1 - sum(x*y) / (norma(x) * norma(y))
}
```

Y calculamos todas las posibles distancias (normalmente **no** queremos hacer esto, pero lo hacemos aquí para comparar):

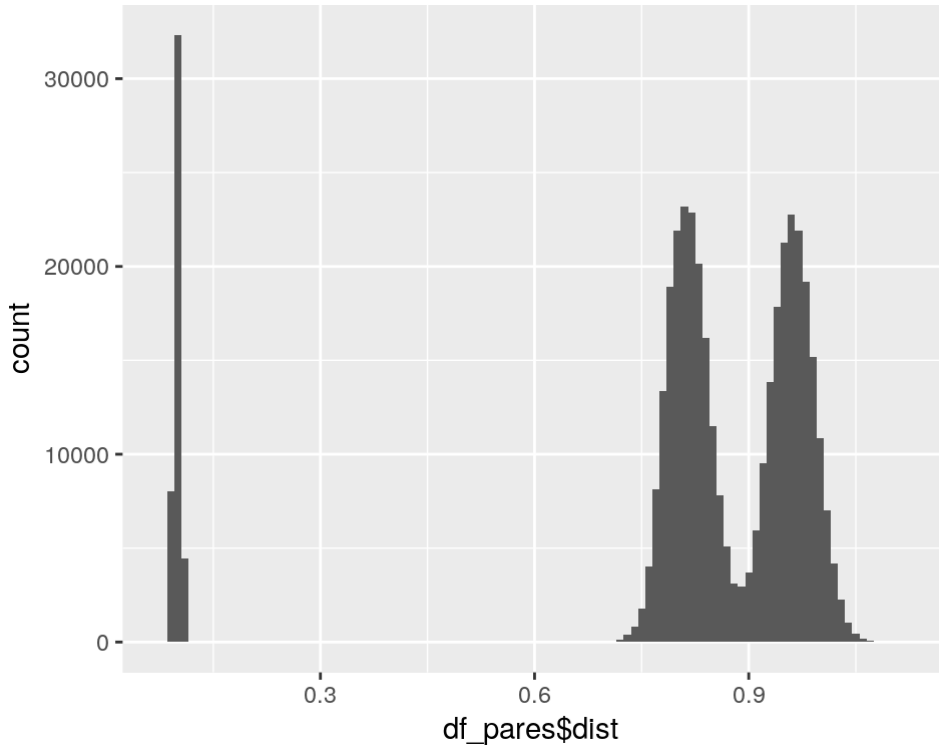
```
df_agrup <- df %>% gather('variable', 'valor', -id_1) %>%
                   group_by(id_1) %>%
                   arrange(variable) %>%
                   summarise(vec_1 = list(valor))
df_pares <- df_agrup %>%
            crossing(df_agrup %>%
                       rename(id_2 = id_1, vec_2 = vec_1)) %>%
            filter(id_1 < id_2) %>%
            mutate(dist = map2_dbl(vec_1, vec_2, dist_coseno))
df_pares
```

```
## # A tibble: 404,550 x 5
##     id_1 vec_1          id_2 vec_2           dist
##    <int> <list>        <int> <list>         <dbl>

##  1     1 <dbl [1,000]>     2 <dbl [1,000]> 0.101
##  2     1 <dbl [1,000]>     3 <dbl [1,000]> 0.103
##  3     1 <dbl [1,000]>     4 <dbl [1,000]> 0.0972
##  4     1 <dbl [1,000]>     5 <dbl [1,000]> 0.0970
##  5     1 <dbl [1,000]>     6 <dbl [1,000]> 0.102
##  6     1 <dbl [1,000]>     7 <dbl [1,000]> 0.0981
##  7     1 <dbl [1,000]>     8 <dbl [1,000]> 0.0891
##  8     1 <dbl [1,000]>     9 <dbl [1,000]> 0.0940
##  9     1 <dbl [1,000]>    10 <dbl [1,000]> 0.0957
## 10     1 <dbl [1,000]>    11 <dbl [1,000]> 0.102
## # ... with 404,540 more rows
```

La distribución de distancias sobre todos los pares es la siguiente: (¿por qué observamos este patrón? Recuerda que esta gráfica representa pares):

```
qplot(df_pares$dist, binwidth = 0.01)
```



Y supongamos que queremos encontrar vectores con distancia coseno menor a 0.2 (menos de unos 40 grados). El número de pares que satisfacen esta condicion son:

```
sum(df_pares$dist < 0.20)
```

```
## [1] 44850
```

Ejemplo: LSH planos aleatorios

Con 200 funciones hash:

```
set.seed(101021)
hashes <- lapply(1:200, function(i){
    v <- rnorm(1000)
    function(x){
        ifelse(sum(v*x) >= 0, 1, -1)
    }
})
```

Por ejemplo, la firma del primer elemento es:

```
x <- as.numeric(df[1,-1])
sapply(hashes, function(f) f(x))
```

```
##   [1] -1  1  1 -1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1 -1  1  1  1  1 -1  1
##  [24] -1  1  1 -1  1  1 -1 -1 -1  1 -1  1  1  1 -1 -1  1 -1  1 -1  1  1  1
##  [47]  1 -1  1  1 -1 -1 -1  1  1  1  1  1 -1  1 -1  1  1 -1 -1  1 -1  1 -1
##  [70] -1  1  1 -1  1 -1  1 -1  1  1 -1 -1 -1 -1 -1  1 -1  1  1 -1  1 -1  1
##  [93] -1  1  1  1 -1 -1  1  1 -1  1  1  1 -1 -1  1  1  1 -1  1  1 -1  1  1
## [116] -1 -1  1 -1 -1 -1 -1  1 -1  1  1  1 -1  1 -1 -1 -1 -1 -1 -1  1  1  1
## [139]  1 -1  1 -1  1  1 -1 -1 -1 -1 -1  1  1  1  1  1  1 -1  1  1  1 -1 -1
## [162] -1  1  1 -1  1 -1  1 -1  1  1  1  1  1 -1 -1 -1  1  1 -1  1 -1 -1  1
## [185] -1 -1 -1  1  1 -1  1  1 -1  1 -1 -1  1 -1 -1 -1
```

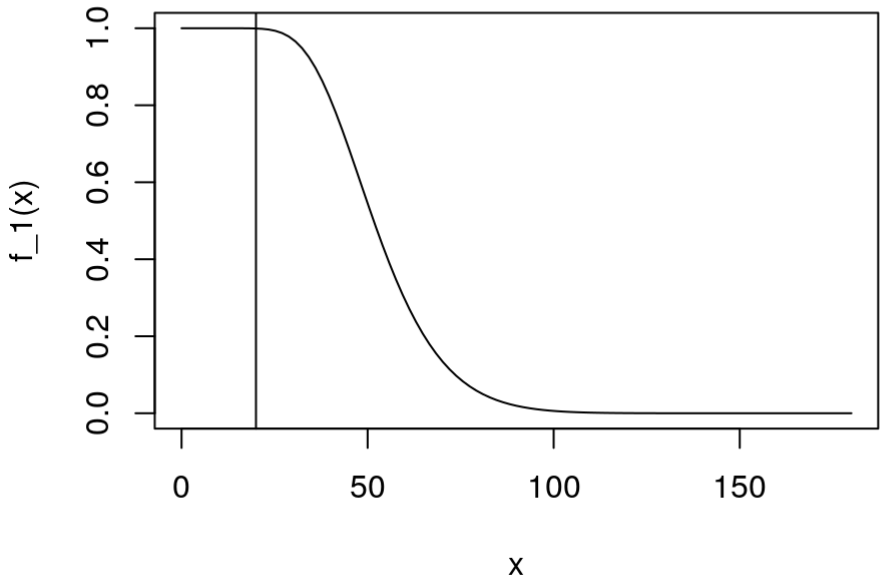
Y ahora calcuamos la firma para cada elemento:

```
df_hash <- df_agrup %>%
           mutate(df = map(vec_1, function(x){
              firma <-  sapply(hashes, function(f) f(x))
              data_frame(id_hash = 1:length(firma),
                         firma = firma) })) %>%
              select(-vec_1) %>% unnest
df_hash
```

```
## # A tibble: 180,000 x 3
##     id_1 id_hash firma
##    <int>   <int> <dbl>
##  1     1       1 -1.00
##  2     1       2  1.00
##  3     1       3  1.00
##  4     1       4 -1.00
##  5     1       5  1.00
##  6     1       6 -1.00
##  7     1       7 -1.00
##  8     1       8 -1.00
##  9     1       9 -1.00
## 10     1      10 -1.00
## # ... with 179,990 more rows
```

Vamos a amplificar la famiia de hashes. En este caso, escogemos 20 bandas de 10 bandas cada una.

```
f_1 <- function(x){
    1-(1-((180-x)/180)^10)^20
}
curve(f_1, 0, 180)
abline(v=20)
```



Ejemplo: agrupar por cubetas para LSH

Ahora agrupamos y construimos las cubetas:

```
df_hash_1 <- df_hash %>%
            mutate(banda  = (id_hash - 1) %% 20 + 1) %>%
            mutate(h = paste(id_hash,firma)) %>%
            arrange(id_1)
df_hash_1
```

```
## # A tibble: 180,000 x 5
##     id_1 id_hash firma banda h
##    <int>   <int> <dbl> <dbl> <chr>
##  1     1       1 -1.00  1.00 1 -1
##  2     1       2  1.00  2.00 2 1
##  3     1       3  1.00  3.00 3 1
##  4     1       4 -1.00  4.00 4 -1
##  5     1       5  1.00  5.00 5 1
##  6     1       6 -1.00  6.00 6 -1
##  7     1       7 -1.00  7.00 7 -1
##  8     1       8 -1.00  8.00 8 -1
##  9     1       9 -1.00  9.00 9 -1
## 10     1      10 -1.00 10.0  10 -1
## # ... with 179,990 more rows
```

```
cubetas <- df_hash_1 %>%
             group_by(id_1, banda) %>%
             summarise(cubeta = paste(h, collapse = '/'))
cubetas
```

```
## # A tibble: 18,000 x 3
## # Groups:   id_1 [?]
##     id_1 banda cubeta
##    <int> <dbl> <chr>
##  1     1  1.00 1 -1/21 1/41 1/61 1/81 -1/101 -1/121 -1/141 1/161 -1/181 1
##  2     1  2.00 2 1/22 -1/42 1/62 -1/82 -1/102 1/122 -1/142 -1/162 -1/182 1
##  3     1  3.00 3 1/23 1/43 1/63 1/83 -1/103 -1/123 -1/143 1/163 -1/183 1
##  4     1  4.00 4 -1/24 -1/44 -1/64 -1/84 -1/104 -1/124 -1/144 1/164 1/184…
##  5     1  5.00 5 1/25 1/45 1/65 -1/85 1/105 -1/125 1/145 1/165 1/185 -1
##  6     1  6.00 6 -1/26 1/46 -1/66 -1/86 -1/106 -1/126 1/146 -1/166 1/186 …
##  7     1  7.00 7 -1/27 -1/47 1/67 1/87 1/107 1/127 1/147 -1/167 -1/187 -1
##  8     1  8.00 8 -1/28 1/48 1/68 1/88 1/108 1/128 -1/148 -1/168 1/188 1
##  9     1  9.00 9 -1/29 1/49 1/69 -1/89 -1/109 1/129 1/149 1/169 -1/189 1
## 10     1 10.0  10 -1/30 -1/50 1/70 -1/90 1/110 1/130 -1/150 1/170 1/190 -1
## # ... with 17,990 more rows
```

```
cubetas_hash <- cubetas %>%
                ungroup %>% rowwise %>%
                mutate(cubeta = digest::digest(cubeta))
cubetas_hash
```

```
## Source: local data frame [18,000 x 3]
## Groups: <by row>
##
## # A tibble: 18,000 x 3
##     id_1 banda cubeta
##    <int> <dbl> <chr>
##  1     1  1.00 52d6b7cbaa37a294cf03b190ff7355c1
##  2     1  2.00 fa72760856b42777f6cf5eb87d544ba2
##  3     1  3.00 f93997cb9c6ad35735a7cb272ee5c4ac
##  4     1  4.00 180f9889f38f66e91e0cc0982b664a5e
##  5     1  5.00 cc5957d8b1f1282d5e141ed1d1689833
##  6     1  6.00 8ca60c23dd2c9a1485d1378345aaf7b4
##  7     1  7.00 c4f05677549fc7ebec213339657218a0
##  8     1  8.00 3ed62cda331157e3bc4f0d10675c11e8
##  9     1  9.00 720aaa0179bbdd052d7f7fdc4f3cb2f6
## 10     1 10.0  4188122161a8cce37edfba3b0b091ef8
## # ... with 17,990 more rows
```

```
cubetas_agrup <- cubetas_hash %>% group_by(cubeta) %>%
                  summarise(ids = list(id_1)) %>%
                  mutate(num_ids = map_dbl(ids, length)) %>%
                  filter(num_ids > 1 )
```

```
## Warning: Grouping rowwise data frame strips rowwise nature
```

```
cubetas_agrup
```

```
## # A tibble: 2,884 x 3
##    cubeta                           ids        num_ids
##    <chr>                            <list>       <dbl>
##  1 0008259d34e4efde7f3317a13a014b57 <int [2]>     2.00
##  2 0032a3ed33002aa835e67b6c97da45d6 <int [2]>     2.00
##  3 0040661be045f0e92548c377d6eb3bcd <int [3]>     3.00
##  4 00628c8e86bd0eb938aca9e60fdf7d74 <int [2]>     2.00
##  5 0066291e6f73898b35d76b55df95441e <int [2]>     2.00
##  6 0077b7483dfa7395ac7df1ae292f7f65 <int [2]>     2.00
##  7 0095793aac09a69b9e437344d05007fc <int [71]>   71.0
##  8 0099e5e147a2853d555c5df7a01c3ffe <int [3]>     3.00
##  9 00ae38f9f1c043d68784c4e5a8a397d5 <int [2]>     2.00
## 10 00b211989c4a1b1398c319ba273a7fc5 <int [2]>     2.00
## # ... with 2,874 more rows
```

Y ahora extraemos los pares similares

```
pares_candidatos <- lapply(cubetas_agrup$ids, function(x){
  combn(sort(x), 2, simplify = FALSE)}) %>%
  flatten %>% unique %>%
  transpose %>% lapply(as.integer) %>% as.data.frame
names(pares_candidatos) <- c('id_1','id_2')
head(pares_candidatos)
```

```
##   id_1 id_2
## 1  752  812
## 2  674  688
## 3  304  341
## 4  304  496
## 5  341  496
## 6  735  835
```

Ejemplo: filtrar y evaluar resultados

Y ahora evaluamos nuestros resultados. En primer lugar, el número de pares reales y de candidatos es

```
pares_reales <- filter(df_pares, dist < 0.15) %>%
                select(id_1, id_2)
nrow(pares_reales)
```

```
## [1] 44850
```

```
nrow(pares_candidatos)
```

```
## [1] 58025
```

Así que debemos tener buen número de falsos positivos. Podemos calcularlos haciendo

```
nrow(anti_join(pares_candidatos, pares_reales))
```

```
## Joining, by = c("id_1", "id_2")
```

```
## [1] 14182
```

Y el número de falsos negativos es

```
nrow(anti_join(pares_reales, pares_candidatos))
```

```
## Joining, by = c("id_1", "id_2")
```

```
## [1] 1007
```

que es un porcentaje bajo del total de pares reales.

**Observación*: es posible, en lugar de usar vectores con dirección aleatoria v escogidos al azar como arriba (con la distribución normal), hacer ménos cálculos escogiendo vectores v cuyas entradas son solamente 1 y -1. El cálculo del producto punto es simplemente multiplicar por menos si es necesario los valores de los vectores x y sumar.

3.11 LSH para distancia euclideana.

Para distancia euclideana usamos el enfoque de proyecciones aleatorias en cubetas.

La idea general es que tomamos una línea al azar en el espacio de entradas, y la dividimos en cubetas de manera uniforme. El valor hash de un punto x es el número de cubeta donde cae la proyección de x.

> Supogamos que tomamos como a el ancho de las cubetas. La familia de proyecciones aleatorios por cubetas es una familia $(a/2, 2a, 1/2, 1/3)$-sensible a la localidad para la distancia euclideana.

Supongamos que dos punto x y y tienen distancia euclideana $d = a/2$. Si proyectamos perpendicularmente sobre la línea escogida al azar, la distancia entre las proyecciones es menor a $a/2$, de modo la probabilidad de que caigan en la misma cubeta es al menos 1/2. Si la distancia es menor, entonces la probabilidad es más grande aún:

1. Si $d(x,y) \leq a/2$ entonces $P(f(x) = f(y)) \geq 1/2$.

Por otro lado, si la distancia es mayor a $2a$, entonces la única manera de que los dos puntos caigan en una misma cubeta es que la distancia de sus proyecciones sea menor a a. Esto sólo puede pasar si el ángulo entre el vector que va de x a y y la línea escogida al azar es mayor de 60 a 90 grados. Como $\frac{90-60}{90-0} = 1/3$, entonces la probabilidad que que caigan en la misma cubeta no puede ser más de 1/3.

1. Si $d(x,y) \geq 2a$ entonces $P(f(x) = f(y)) \leq 1/3$.

Escoger a para discriminar las distancias que nos interesa, y luego amplificar la familia para obtener tasas de falsos positivos y negativos que sean aceptables.

Tarea

1. La tarea está en *scripts/lsh/entity-matching.Rmd*
2. Si quieres hacer experimentos con el error en el cálculo de similitud por minhashing (y la diferencia entre permutaciones y hashes), puedes ver el script *scripts/lsh/notas_adicionales_minhash.nb.html* (o correr el Rmd asociado).

References

Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets*. 2nd ed. New York, NY, USA: Cambridge University Press.

Mullen, Lincoln. 2016. *Textreuse: Detect Text Reuse and Document Similarity*. https://CRAN.R-project.org/package=textreuse.