

# Recuperación de Información (RI) sobre texto

Sistemas Inteligentes  
de Acceso a la Información  
José María Gómez Hidalgo

<http://www.esi.uem.es/~jmgomez/sinai/>

# Índice

- Modelos clásicos
- El modelo booleano
- El Modelo del Espacio Vectorial (MEV)
- Comparación de modelos
- Implementación
- Análisis léxico
- Esquema de indexación automática

# Modelos clásicos

- Tres tipos de modelos
  - Modelo booleano (basado en conjuntos)
  - Modelo del Espacio Vectorial (algebraico)
  - Modelo probabilístico
- Son la base de muchos modelos avanzados
- Perviven en muchas aplicaciones

# Modelos clásicos

- En todos ellos se asume que los documentos están caracterizados por
  - Una serie de términos índice o palabras clave (índices) e.g. “perro”, “perr”, “perro lobo”  
 $T = \{t_1, \dots, t_N\}, |T| = N$
  - Una función que indica la “importancia” de cada término en cada documento  
 $D = \{d_1, \dots, d_M\}, |D| = M$   
 $F: T \times D \rightarrow R, F(t_i, d_j) = w_{ij}, \text{ e.g. } w_{ij} = 1$

# Modelos clásicos

- Se asume la consulta  $C$  caracterizada por mismos términos índice, con su grado de importancia  
 $F(C): T \rightarrow R$ , e.g.  $F(C,t_i) = wc_i = 1 \Leftrightarrow t_i \text{ en } C$
- Se pretende caracterizar la “relevancia” con una función de similitud  
 $SIM(C): D \rightarrow [0, 1]$ ,  $SIM(C,d_j)$  aproxima el grado de relevancia de  $d_j$  a  $C$  (e.g. = 1 si es máximo)

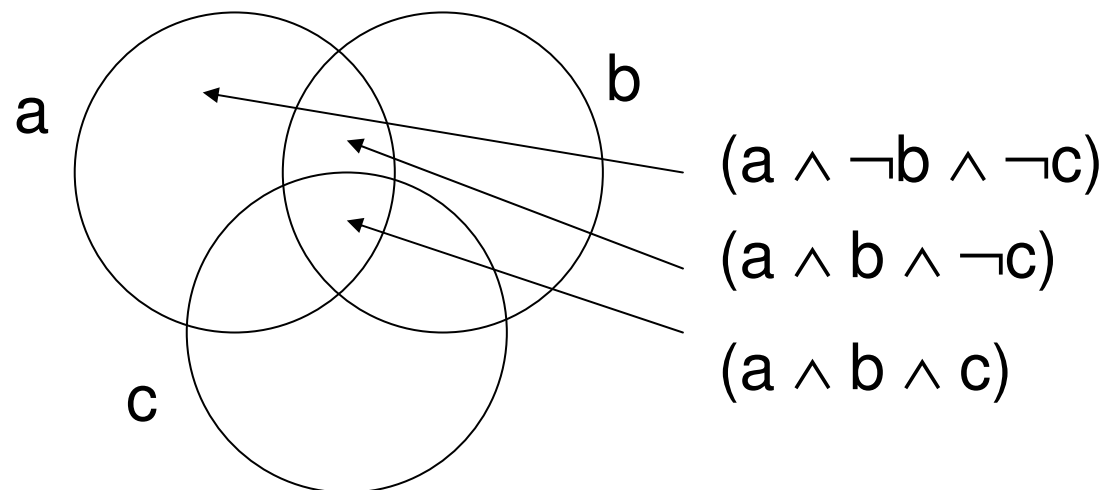
# El modelo booleano

- Sencillo, basado en teoría de conjuntos y algebra de Boole (operadores en consulta)
- Ventajas
  - Sencillez, elegancia
  - Semántica de expresiones booleanes
- Desventajas
  - No hay graduación (*ranking*) = relevancia 0 ó 1
  - Usar operadores booleanos ( $\neg$ ,  $\wedge$ ,  $\vee$ ) no es trivial

# El modelo booleano

- Expresión booleana  $\Rightarrow$  forma normal disyuntiva (DNF, *disjunctive normal form*)

$$a \wedge (b \vee \neg c) \Leftrightarrow (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c)$$



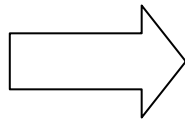
# El modelo booleano

- <http://www.everything2.com/> => “disjunctive normal form”
- Toda función booleana se puede expresar en DNF => el conjunto de conectivas  $\{\neg, \wedge, \vee\}$  es funcionalmente completo
- Algoritmo simple = análisis de tabla de verdad (componentes DNF exponencial  $O(2^n)$  siendo  $n$  el número de literales distintos)

# El modelo booleano

- E.g. Dada  $f(p,q,r)$ ,  $f: \Sigma^3 \rightarrow \{0,1\}$ , y su tabla de verdad...

p	q	r		$f(p,q,r)$
0	0	0		1
0	0	1		1
0	1	0		0
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		0
1	1	1		0



p	q	r		$f(p,q,r)$	
0	0	0		1	$(\neg p \wedge \neg q \wedge \neg r) \vee$
0	0	1		1	$(\neg p \wedge \neg q \wedge r) \vee$
0	1	0		0	
0	1	1		0	
1	0	0		1	$(p \wedge \neg q \wedge \neg r)$
1	0	1		0	
1	1	0		0	
1	1	1		0	

# El modelo booleano

- Cálculo de similitud

$SIM(C): D \rightarrow [0,1]$

Sea  $C^* = c_1 \vee \dots \vee c_K$ , siendo cada  $c_k$  una secuencia de conjunciones, y  $d_j^*$  = una conjunción restringida a los términos de la consulta (los demás se ignoran) para el documento  $d_j$

$SIM(C, d_j) = 1 \Leftrightarrow (c_1 \wedge d_j^*) \vee \dots \vee (c_K \wedge d_j^*)$ , 0 e.o.c

Es decir, la similitud es 1 si se satisface simultáneamente alguna conjunción de la consulta, y la conjunción del documento

# El modelo booleano

- Cálculo de similitud

$$\Sigma = T = \{a, b, c, d, e\}, C^* = (a \wedge \neg b) \vee (\neg a \wedge b)$$

$$d_1 = (\neg a \wedge b \wedge \neg c \wedge d \wedge \neg e) \Rightarrow d_1^* = (\neg a \wedge b)$$

$$d_2 = (\neg a \wedge \neg b \wedge c \wedge d \wedge \neg e) \Rightarrow d_2^* = (\neg a \wedge \neg b)$$

$\Rightarrow$

$$\text{SIM}(C, d_1) = 1 \text{ (se tiene } (c_2 \wedge d_1^*))$$

$$\text{SIM}(C, d_2) = 0 \text{ (no se tiene } (c_1 \wedge d_2^*) \text{ ni } (c_2 \wedge d_2^*))$$

Obsérvese la decisión *binaria*

# El Modelo del Espacio Vectorial

- Más sofisticado que el booleano
- Basado en álgebra vectorial
- Permite ajuste parcial y *ranking*
- Representación de expresión en lenguaje natural como *vector de pesos de términos*
  - Pesos = binarios, TF (Term Frequency), TF.IDF (Inverse Document Frequency)
  - Similitud = distancia entre puntos

# El Modelo del Espacio Vectorial

- Cálculo de pesos

$d_j = [w_{1j}, \dots, w_{Nj}]$ ,  $w_{ij}$  se puede definir como:

BINARIO –  $w_{ij} = 1$  si  $t_i \in d_j$ , 0 e.o.c. ( $\Rightarrow$  consulta)

TF –  $w_{ij} = tf_{ij} = \text{núm. de veces que } t_i \text{ aparece en } d_j$   
(mayor frecuencia, mayor peso)

TF.IDF –  $w_{ij} = tf_{ij} \cdot idf_i = \text{peso en el documento } x$   
 $\text{peso en la colección, siendo } df_i = \text{núm. docs. con } t_i$

$$idf_i = \log_2 \left( \frac{M}{df_i} \right)$$

# El Modelo del Espacio Vectorial

- Cálculo de similitud

$SIM(C): D \rightarrow [0,1], |T| = N$

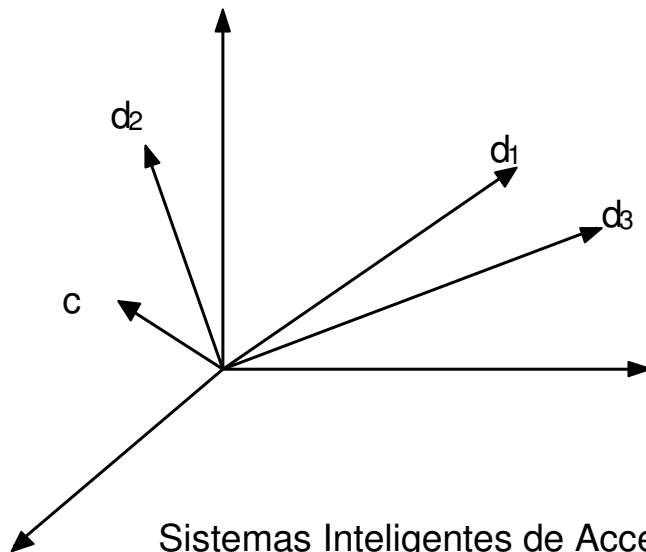
Sea  $C = [wc_1, \dots, wc_N]$ , y sea  $d_j = [w_{1j}, \dots, w_{Nj}]$

$$SIM(C, d_j) = \cos(C, d_j) = \frac{C \cdot d_j}{|C||d_j|} = \frac{\sum_{i=1}^N wc_i \cdot w_{ij}}{\sqrt{\sum_{i=1}^N wc_i^2} \sqrt{\sum_{i=1}^N w_{ij}^2}}$$

Otras similitudes posibles (e.g. inversa euclídea)

# El Modelo del Espacio Vectorial

	algorithm	architecture	computer	logic	program
Documento1	2.23	5.34	2.45	0.00	0.00
Documento2	3.50	0.00	0.00	3.20	1.51
Documento3	0.00	4.76	3.23	0.00	2.31
Consulta	0.00	0.00	0.00	1.06	0.74



similitudes entre documentos y consulta:

$$\text{sim}(d_1, c) = 0.0000$$

$$\text{sim}(d_2, c) = 0.7009$$

$$\text{sim}(d_3, c) = 0.2133$$

# Comparación de modelos

- El modelo booleano se considera el peor
  - Incapacidad de reconocimiento parcial
- Controversia entre MEV y probabilístico
  - Croft evidenció <
  - Salton evidenció >
  - El MEV es más popular en la práctica

# Implementación

- Importante para lograr eficiencia
  - Grandes volúmenes de información
  - Usabilidad en determinados entornos (e.g. Web)
- Orientado a aspectos básicos
  - Excluimos mejoras avanzadas (e.g. caching)
- Referencia
  - C. Faloutsos, D. Oard, 1996. A Survey of Information Retrieval and Filtering Methods. Technical Report, Information Filtering Project, University of Maryland, College Park.

# Implementación

- Idea más básica
  - Dada un término  $w$ , buscar documento  $a$  documento (sujeto a las condiciones booleanas)
    - E.g. se requiere  $w \in d$ , ó bien  $w \notin d$
- Algoritmo secuencial directo, multidireccional
  - Buscar la cadena  $w$  en  $d$ , con una ventana de longitud  $|w|$ , avanzando 1 posición por éxito o fracaso
  - Requiere  $O(|d||w|)$  en tiempo,  $O(1)$  memoria

# Implementación

```
busqueda_ingenua( d, n, w, m )  
/* Busca w[0..m-1] en d[0..n-1] */  
char w[], d[]; int n, m;  
{ int i, j, k, lim;  
  lim = n-m+1;  
  for( i = 0; i < lim; i++ ) /* busqueda */  
  { k = i;  
    for( j=0; j<m && d[k] == w[j]; j++ ) k++;  
    if( j >= m ) exito_en_posicion( i ); } }
```

# Implementación

- Boyer-Moore
  - Mejor algoritmo conocido (salvo pequeñas mejoras)
  - Presente en utilidades de buscar y reemplazar de la mayoría de editores de texto
  - Idea básica: buscar  $w$  de izda a dcha en  $d$ , ajustar de dcha a izda, y salto variable (hasta de  $|w|$  posiciones en  $d$ )
  - Tiempo  $O(|w|+|d|)$  con preprocesamiento en  $O(|w|)$ , y memoria  $O(|w| + |\Sigma|)$  -  $\Sigma =$  alfabeto

# Implementación

- Boyer-Moore simplificado
  - Se calcula el salto por medio de una heurística de aparición
    - Alinear la posición del documento que causó el fallo con el primer carácter de la palabra que encaja con ella
  - Si  $|w| = m$ 
    - $D(x) = \min\{s \mid s=m \text{ ó } (0 \leq s < m \text{ y } w[m-s] = x)\} \forall s \in \Sigma$
  - Se almacena en una tabla D y el valor es el salto
  - Tiempo  $O(|w||d|)$  con preprocesamiento en  $O(|\Sigma|)$ , y memoria  $O(|\Sigma|)$

# Implementación

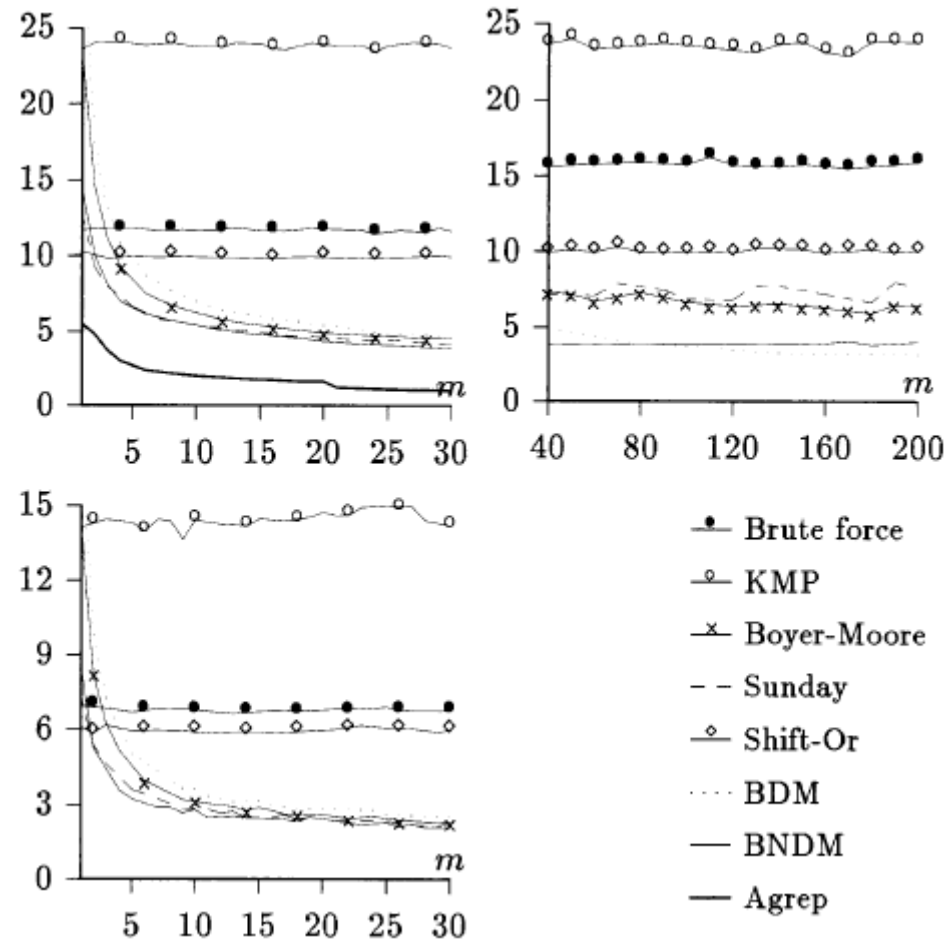
```
initD( w, m, D )  
/* Preprocesar palabra w de tamaño m => tabla D */  
char w[];  
int m, D[];  
{  
    int k;  
    for( k=0; k < MAX_ALPHABET_SIZE; k++ ) D[k] = m;  
    for( k=0; k < m; k++ ) D[w[k]] = m-k-1;  
}
```

# Implementación

```
BMS( d, n, w, m ) /* Busca w[0..m-1] en d[0..n-1] */
char d[], w[]; int n, m;
{ int k, j, D[MAX_ALPHABET_SIZE];
  initD( w, m, D ); /* Preprocesar patron */
  k = m;
  while( k <= n ) /* Buscar */
  { j = m;
    while( j>0 && d[k-1] == w[j-1] ) { j--; k--; }
    if( j == 0 ) { exito_en_posicion( k ); k += m+1; }
    else k += max(D[d[k-1]],1); } }
```

# Implementación

- Comparación
  - Tiempo en decenas de segundo por Mb
  - A = patrones cortos en inglés
  - B = patrones largos en ADN
  - C = patrones cortos en texto aleatorio
  - (BY) Sección 8.5.6



# Implementación

- Boyer-Moore y muchos otros
  - C. Charras, T. Lecroq, Handbook of exact string matching algorithms. <http://www-igm.univ-mlv.fr/~lecroq/string/>, 1997.
  - Frakes, William B.; Baeza-Yates, Ricardo. Information retrieval: data structures & algorithms. Prentice Hall, 1992 (QA76.9 .D35 I54 Biblioteca UEM).
  - Boyer R.S., Moore J.S., 1977, A fast string searching algorithm. Communications of the ACM. 20:762-772.

# Implementación

- En general se buscan EDs que
  - independientemente del tiempo de generación (offline)
  - abaraten el tiempo de búsqueda (online)
- Ejemplos
  - Archivos de signatura
  - Índices directos
  - Índices inversos

# Implementación

- Partimos de  $F: T \times D \rightarrow R$ ,  $F(t_i, d_j) = w_{ij}$
- Esencialmente matriz dispersa  $D \times T$  (donde  $P(w_{ij} = 0) \approx .9$ )

	$t_1$	$t_2$	$\dots$	$t_N$
$d_1$	$w_{11}$	$w_{21}$		$w_{N1}$
$d_2$	$w_{12}$	$w_{22}$		$w_{N2}$
$\vdots$				
$d_M$	$w_{1M}$	$w_{2M}$		$w_{NM}$

# Implementación

- Índice directo (DxT)
  - Para cada  $d_j$ , lista de términos (posiblemente con pesos)

$d_1$	$t_{11}$	$w_{11}$	$t_{12}$	$w_{12}$	$\dots$	$t_{1D1}$	$w_{1D1}$
$d_2$	$t_{21}$	$w_{21}$	$t_{22}$	$w_{22}$	$\dots$	$t_{2D2}$	$w_{2D2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$d_M$	$t_{M1}$	$w_{M1}$	$t_{M2}$	$w_{M2}$	$\dots$	$t_{MDM}$	$w_{MDM}$

# Implementación

- Índice directo (DxT)
  - Complejidad de operaciones
    - Creación  $\Theta(M.N)$
    - Inserción  $O(N)$
    - Consulta  $O(M.N)$  asumiendo búsqueda lineal
      - Aunque búsqueda binaria más eficiente  $O(M.\log(N))$
      - Usualmente, para cada  $d_j$ , el número de términos  $\ll N$  (e.g.  $0.1 \times N$ )
  - Poco eficiente en consulta, pero útil en otras tareas de acceso (agrupamiento, categorización)

# Implementación

- Índice inverso (TxD)
  - Para cada  $t_i$ , lista de documentos (posiblemente con pesos)

$t_1$	$d_{11}$	$w_{11}$	$d_{12}$	$w_{12}$	...	$d_{1T1}$	$w_{1T1}$
$t_2$	$d_{21}$	$w_{21}$	$d_{22}$	$w_{22}$	...	$d_{2T2}$	$w_{2T2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$t_N$	$d_{N1}$	$w_{N1}$	$d_{N2}$	$w_{N2}$	...	$d_{NTN}$	$w_{NTN}$

# Implementación

- Índice inverso (TxD)
  - Complejidad de operaciones
    - Creación  $\Theta(M.N)$
    - Inserción  $O(N)$
    - Consulta  $O(M.|C|)$  asumiendo recorrido y mezcla de listas
      - Usualmente, para cada  $t_i$ , el número de documentos  $\ll M$  (e.g.  $0.1 \times M$ )
  - Muy eficiente en consulta
  - Implementación en herramientas prácticas

# Implementación

- Índice inverso (TxD)
  - E.g. Google
    - L. Page, S. Brin. The anatomy of a large-scale hypertextual web search engine. Proceedings of the 7th International WWW Conference, 107-117, 1998.
  - E.g. implementación en Java - IR.JAR  
(<http://www.cs.utexas.edu/users/mooney/ir-course/>)
  - Optimizaciones (MEV)
    - Índice: para cada d, se guarda la tf, y para cada t, su df
    - Adicional: para cada d, se guarda |d|

# Análisis léxico

- Texto => términos índice
- Algunos aspectos básicos
  - Tratamiento de dígitos, guiones, etc.
  - Eliminación de palabras “vacías”
  - Normalización morfológica

# Análisis léxico

- Cadena ASCII/Unicode => palabras candidatas a términos índice
- Dígitos (números pobre significado, excepto fechas, etc.)
  - Eliminación, mapeado (13 => dd), exp. Regulares (fechas)
- Guiones (state-of-the-art vs. “state of the art”, despacho B-47)
  - Eliminación, excepciones aparte

# Análisis léxico

- Cadena ASCII/Unicode => palabras candidatas a términos índice
- Símbolos de puntuación (e.g. CIA vs. C.I.A.)
  - Eliminación
- Mayúsculas (Bank vs. bank)
  - Usualmente, mapeo a minúsculas, salvo excepciones

# Análisis léxico

- Eliminación de palabras “vacías”
  - Determinadas palabras muy frecuentes y con poco contenido semántico
    - Artículos, pronombres, conjunciones, adverbios ...
    - Las apariciones de las 10 palabras más frecuentes del inglés constituyen el 20-30% de un documento
  - Se suelen incluir en una lista de parada (*stoplist*) para filtrarlos de los documentos
  - Las listas se obtienen a partir de córpora representativos del idioma o a partir de la colección de documentos

# Análisis léxico

- Eliminación de palabras “vacías”
  - E.g. lista clásica para el inglés contiene 425 términos extraídos del Brown Corpus
    - a about above across after again against all almost alone along already ...
  - E.g. una lista para el español puede contener
    - a añadió aún actualmente adelante además afirmó agregó ahí ahora al algún algo ... (351 palabras)

# Análisis léxico

- Normalización morfológica
  - Múltiples palabras son variaciones morfológicas y con un significado esencialmente idéntico
  - Existen algoritmos de extracción de raíces (*stemmers*) que las normalizan a una forma canónica (no necesariamente la raíz)
    - analizar, análisis, analizador ... => “anali”
  - Se basan en
    - autómatas finitos
    - aplicación recursiva de reglas

# Análisis léxico

- Normalización morfológica
  - E.g. Porter stemmer (inglés)
    - 88 reglas agrupadas en 9 conjuntos, aplicación en cadena

```
static RuleList step2_rules[] = {  
    203, "ational", "ate", 6, 2, 0, NULL,  
    204, "tional", "tion", 5, 3, 0, NULL,  
    205, "enci", "ence", 3, 3, 0, NULL,  
    ...  
    000, NULL, NULL, 0, 0, 0, NULL};
```

# Análisis léxico

- Normalización morfológica
  - E.g. Porter stemmer (castellano)
    - 148 reglas agrupadas en 6 conjuntos, aplicación en cadena

```
static RuleList step2_rules[] = {  
    {201, "ización",    "izar", 6, 2, -1, NULL},  
    {202, "cional",    "ción", 5, 3, -1, NULL},  
    {203, "ructor",    "ruir", 5, 3, -1, NULL},  
    ...  
    {000, NULL,        NULL, 0, 0, 0, NULL}};
```

# Análisis léxico

- Normalización morfológica
- Problemas
  - Infrarradicación
    - hope => “hop”
    - hopping => “hopp”
  - Sobrerradicación
    - capital, capitulate, capitol => “capit”

# Esquema de indexación automática

1. Identificar todas las palabras de los documentos
2. Eliminar las palabras más frecuentes usando una lista de parada
3. Usar un extractor de raíces para reducir las palabras restantes términos en forma canónica
4. *Calcular el poder de discriminación de cada término*

## Esquema de indexación automática

5. *Usar clases de un thesaurus para reemplazar los términos poco frecuentes*
6. *Usar un algoritmo de creación de frases para agrupar los términos de alta frecuencia*
7. Calcular el peso de cada término
8. Asignar a cada documento los pesos de los términos correspondientes