



# IIC3670 Procesamiento de Lenguaje Natural

<https://github.com/marcelomendoza/IIC3670>

## - OUTLINE -

¿Qué vamos a ver?

Ranking y clasificación de documentos

Word2vec, Glove, FastText, Flair

Transformer, BERT, RoBERTa

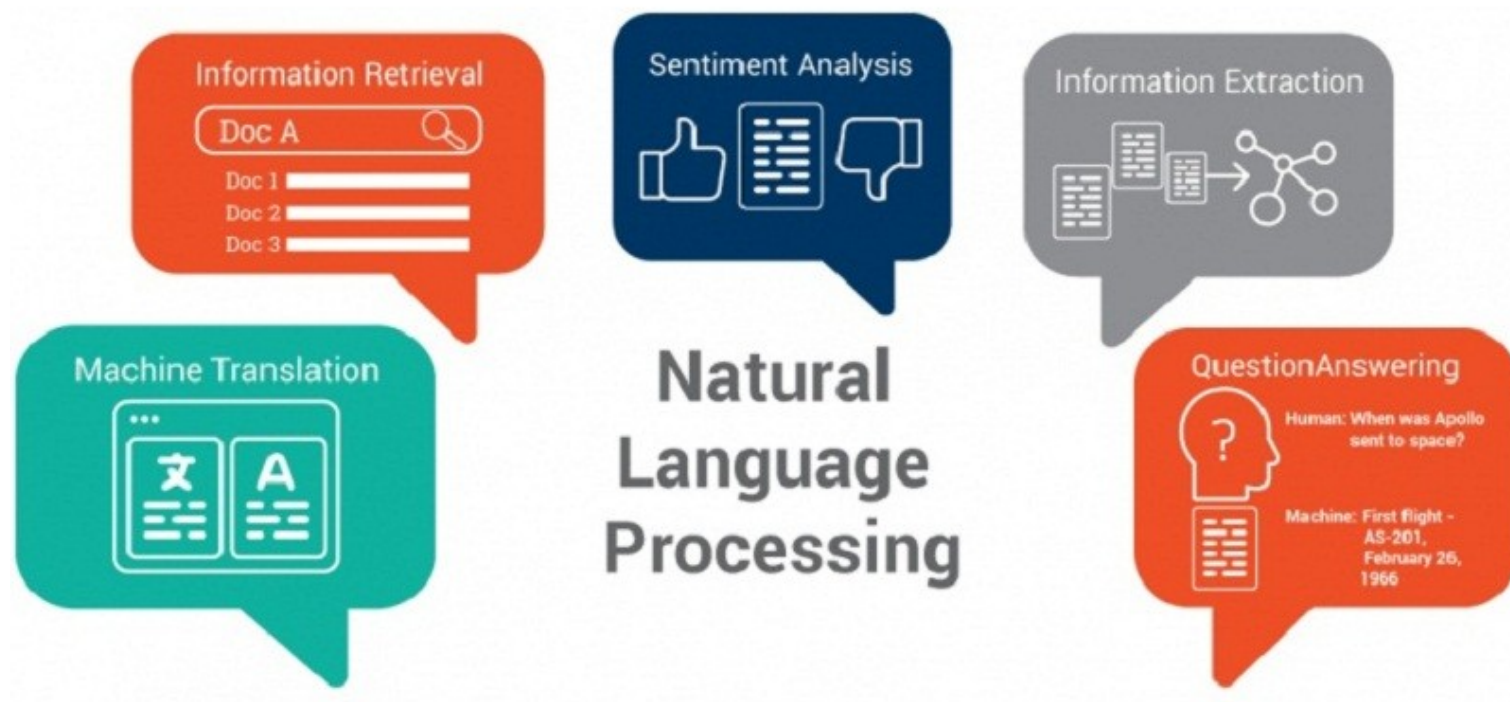
GPT, ChatGPT, alignment

COT, halucinaciones, fairness

# - INTRODUCCIÓN -

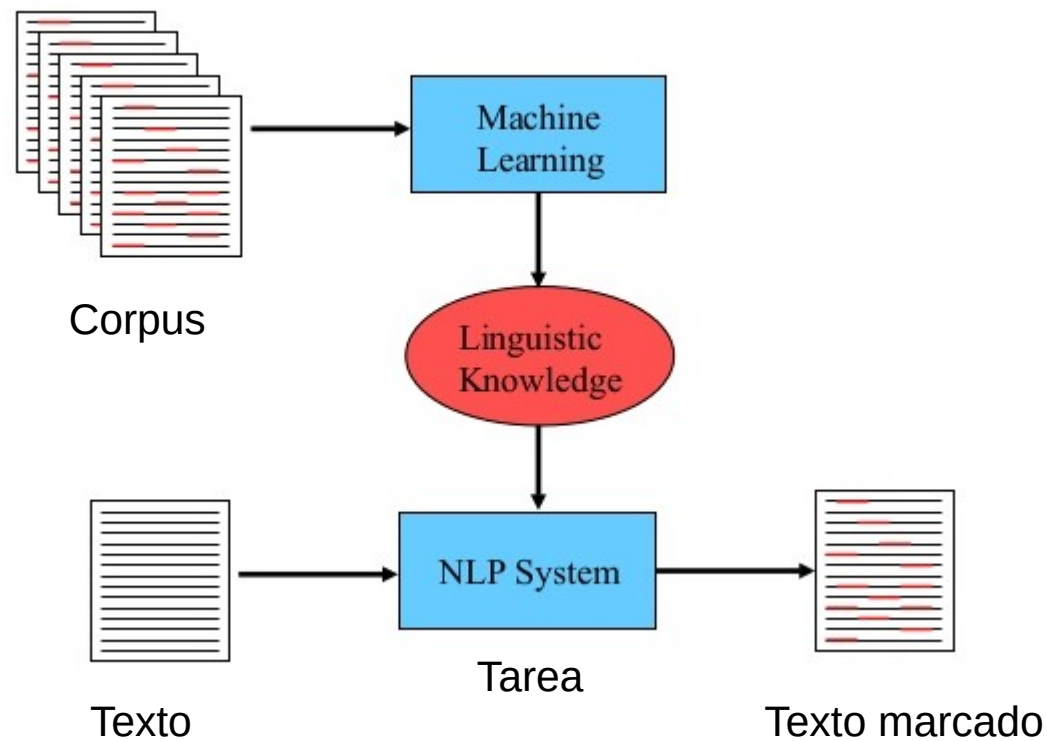
# Introducción

Síntesis. ¿Cuáles tareas aborda NLP?



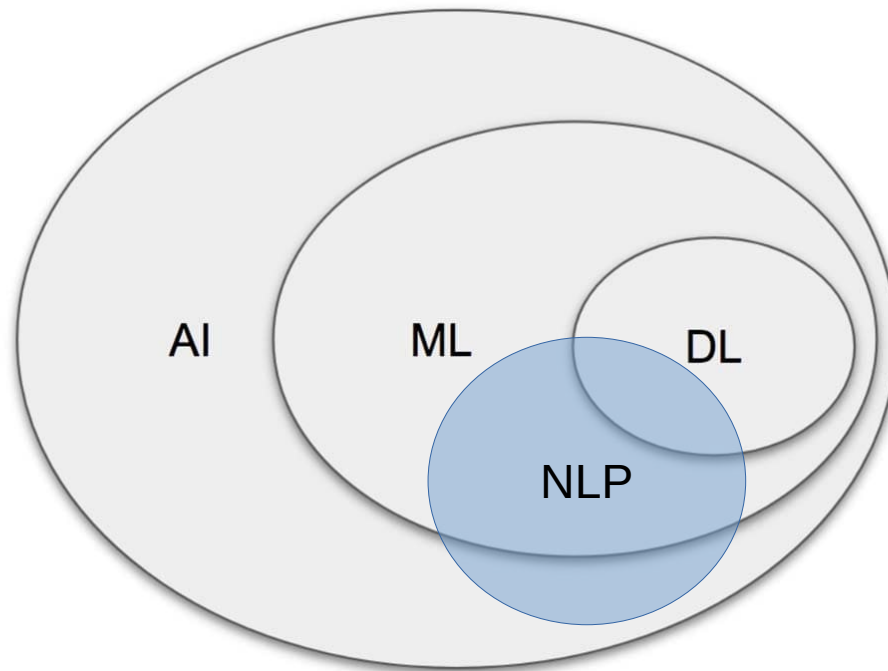
# Introducción

## El enfoque de NLP (clásico)



# Introducción

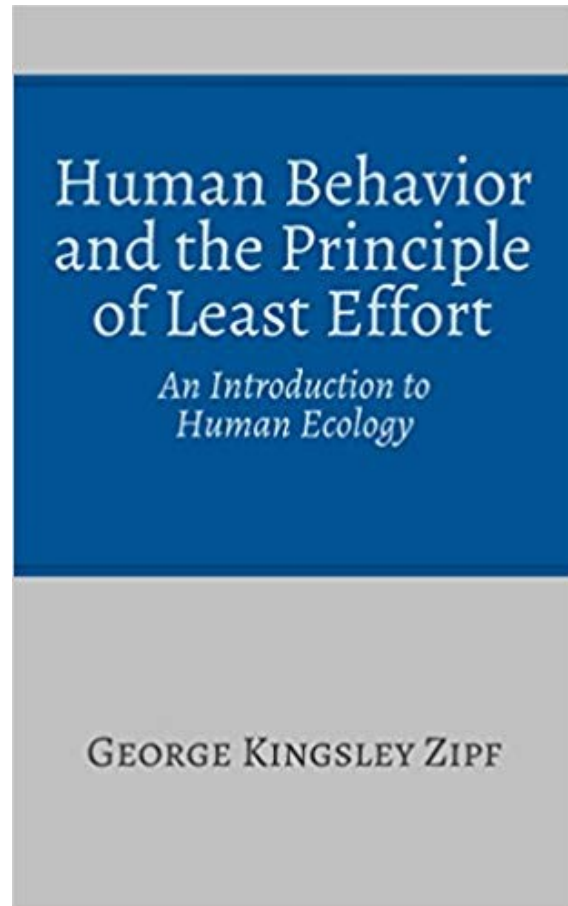
¿Dónde se ubica NLP?



- LEYES DEL TEXTO Y PROCESAMIENTO BÁSICO -



## Leyes del texto



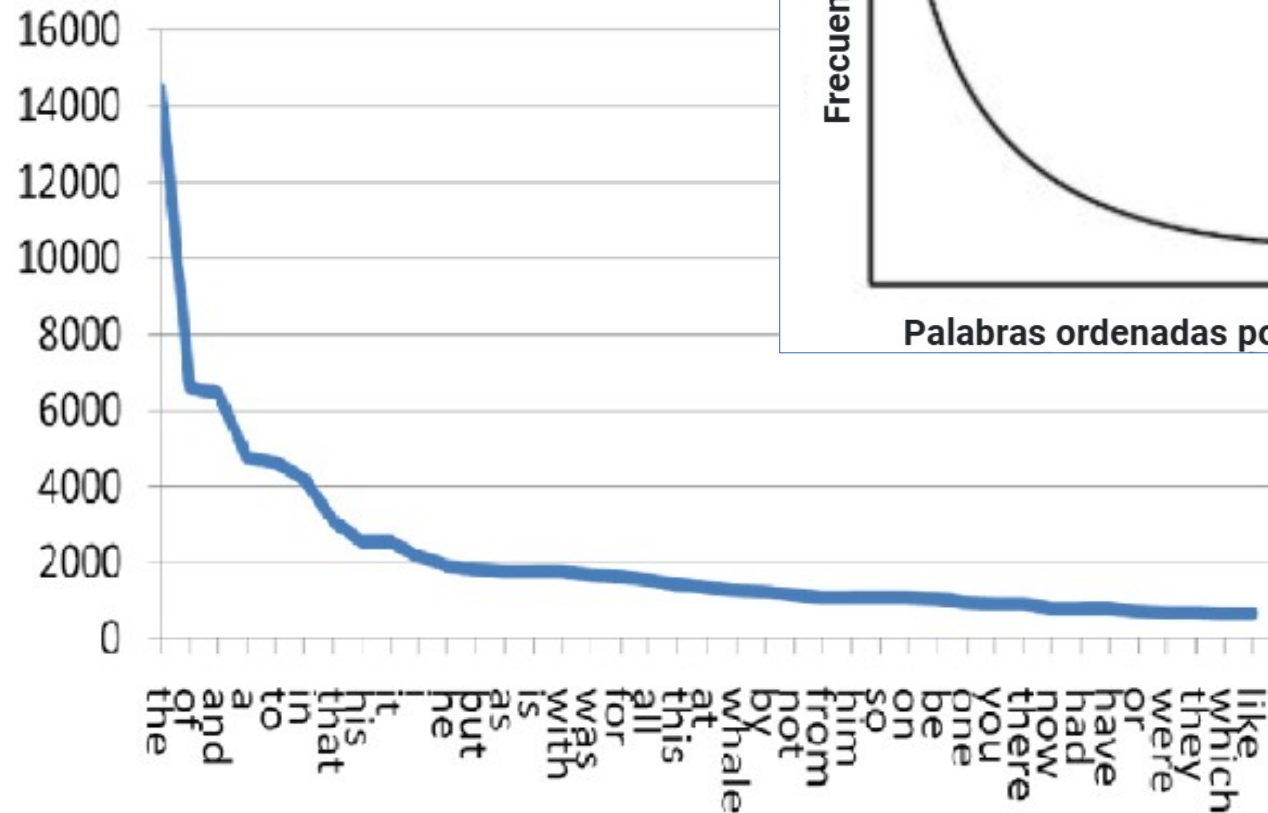
Am I the only one around here that tries to do things with the least effort possible and expects a good result?!



George Kingsley Zipf (1949), Human behavior and the principle of least effort, Addison-Wesley Press

## Leyes del texto

### Zipf para Reuters<sup>2</sup>



<sup>2</sup>Dataset de noticias, disponible on-line

## Leyes del texto

Ley de Zipf:

$$f \sim \frac{1}{r}$$



$$f \sim \frac{1}{r^\theta}$$



$$f_r = \frac{n}{r^\theta \cdot H_V(\theta)}$$

$\theta$  : pendiente de la curva log-log

$n$  : # tokens

$r$  : ranking de la palabra

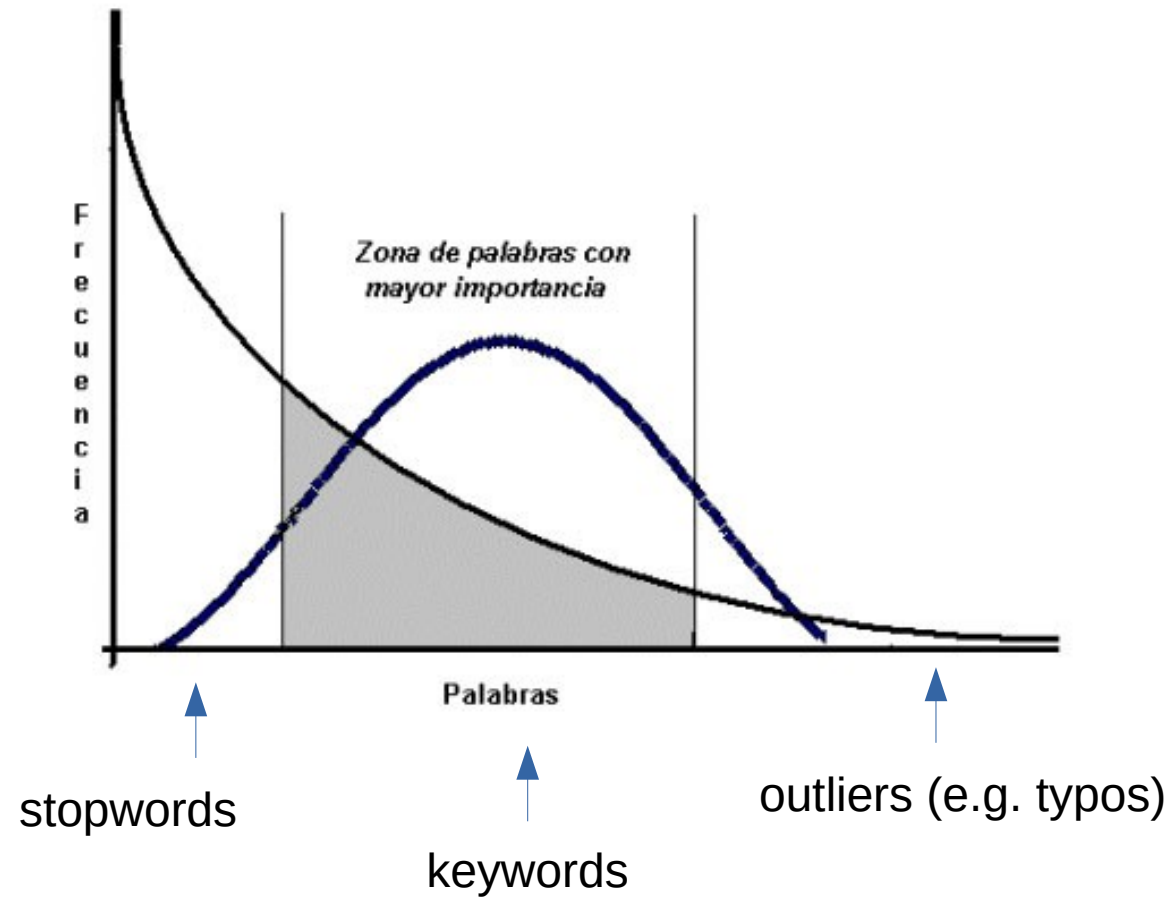
$f$  : # ocurrencias de la palabra

Si  $\theta \approx 1 \rightarrow H_V(\theta) = \log(n)$

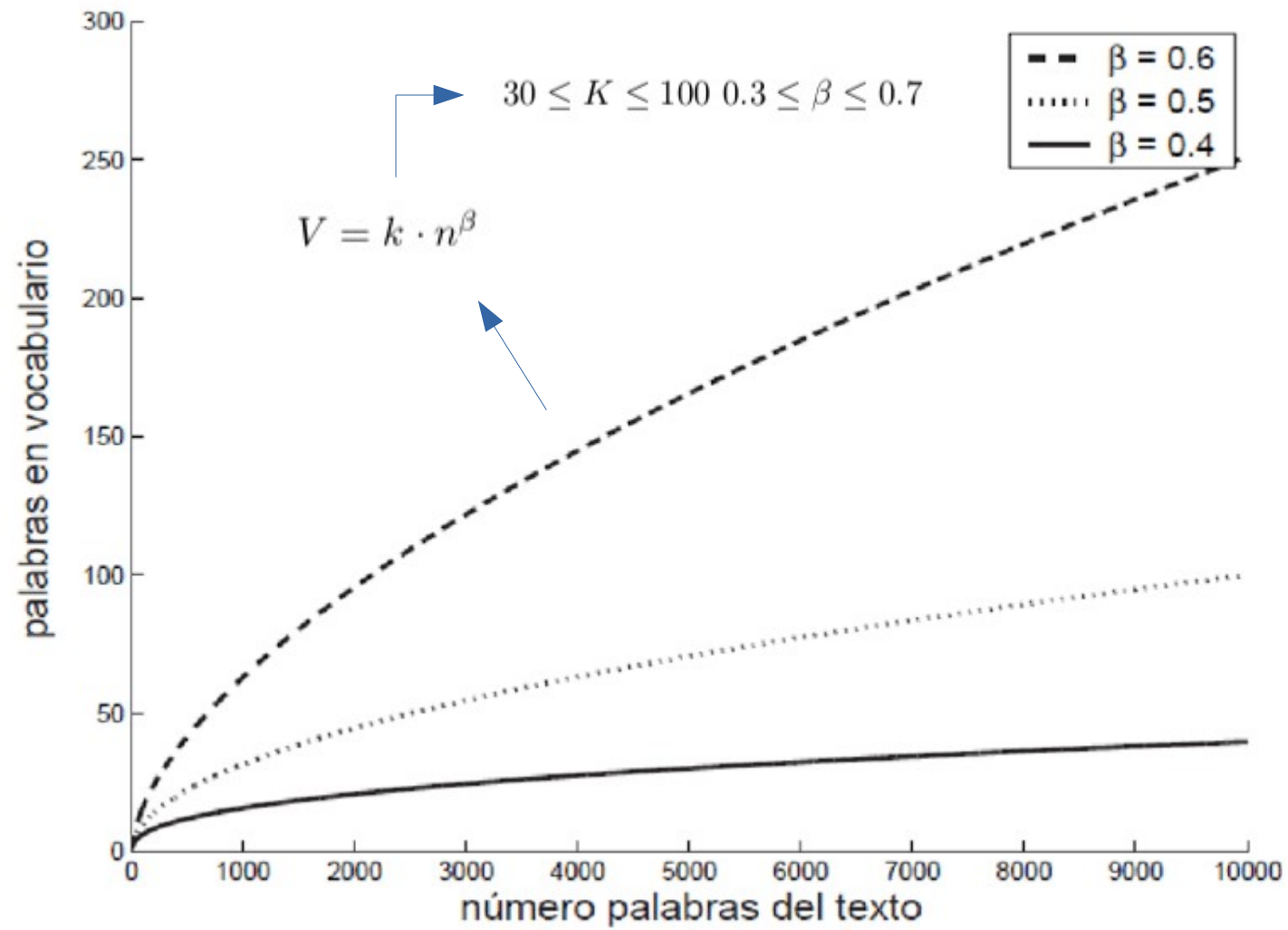


$$H_V(\theta) = \sum_{j=1}^V \frac{1}{j^\theta}$$

## Leyes del texto



## Leyes del texto (Heaps)



## Procesamiento básico

- ▶ **Token** – String delimitado que aparece en el texto.
- ▶ **Término** – token con significado según un corpus (por ejemplo diccionario)
- ▶ **Input:**  

amigos, Romans, habitantes.	habia una vez ... Cesar	...
-----------------------------	-------------------------	-----
- ▶ **Output:**  

amigo	romano	habitante	cesar	...
-------	--------	-----------	-------	-----
- ▶ Cada token es candidato a término.
- ▶ Cuáles elegimos? Depende del corpus.

## Niveles de abstracción del lenguaje

Nivel de la lengua	Unidad de estudio	Disciplina
<b>Morfológico</b>	palabra	Morfología
<b>Sintáctico</b>	oración	Sintaxis
<b>Léxico-semántico</b>	significado	Lexicología y semántica
<b>Textual</b>	texto	Analítica textual

## Monema

La unidad más pequeña dotada de **significante** y **significado**

1

Por ejemplo, la palabra perro **tiene dos monemas**, perr, que denota el significado, y o que denota una cualidad.

2

Los monemas de significado se denominan **lexemas** mientras que los de cualidad se denominan **morfemas**.

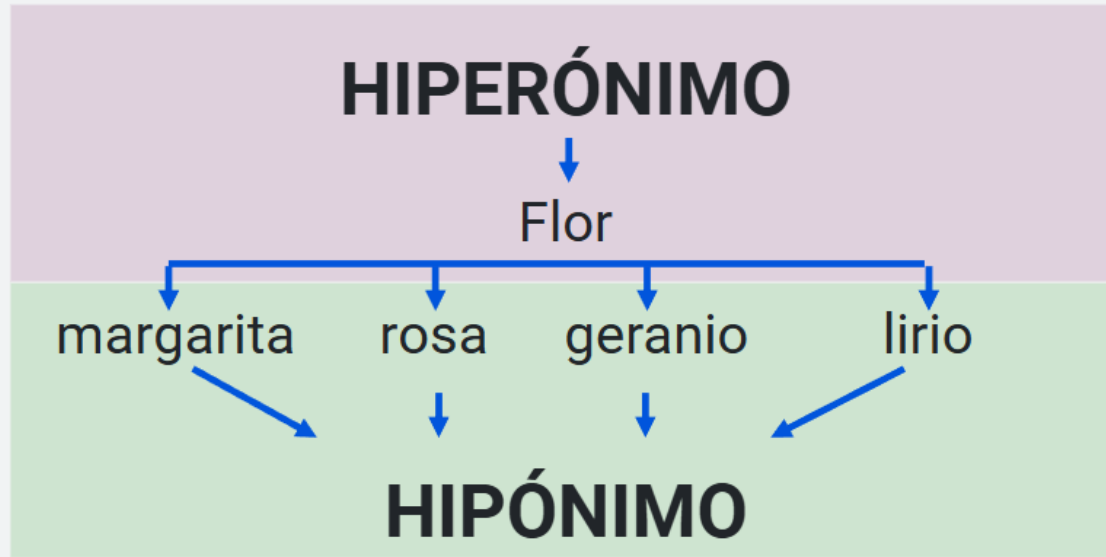




## Variables e invariables



## Hiperonimia o hiponimia



## Procesamiento básico

### Lematización

- ▶ Reducir formas infleccionales a su raíz → Raíz semántica
- ▶ Ejemplo: *am, are, is* → *be*
- ▶ Ejemplo: *autos, auto, automoviles* → *auto*
- ▶ Ejemplo: *Los autos de los jóvenes son de colores* → *auto joven es color*
- ▶ Lematización implica realizar una reducción hacia la raíz (lema).  
(*destruccion* → *destruir*)

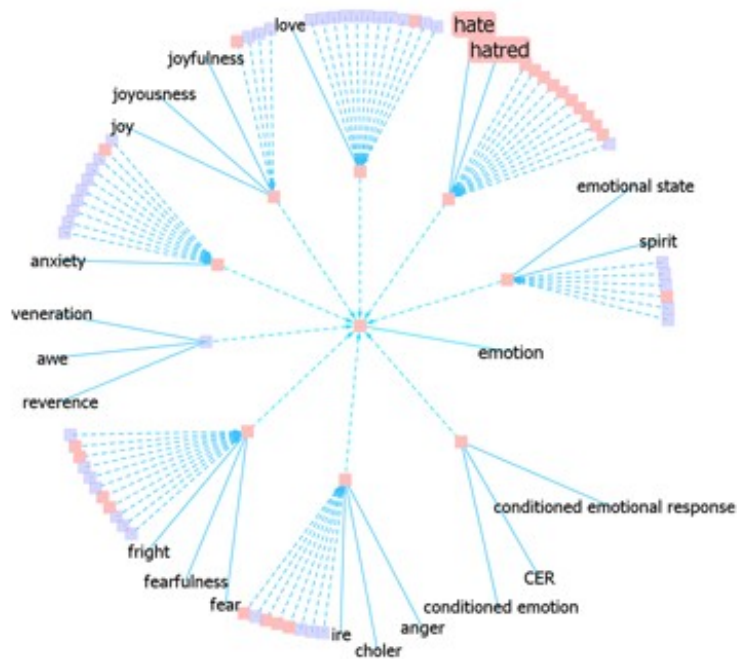
WordNet lemmatizer



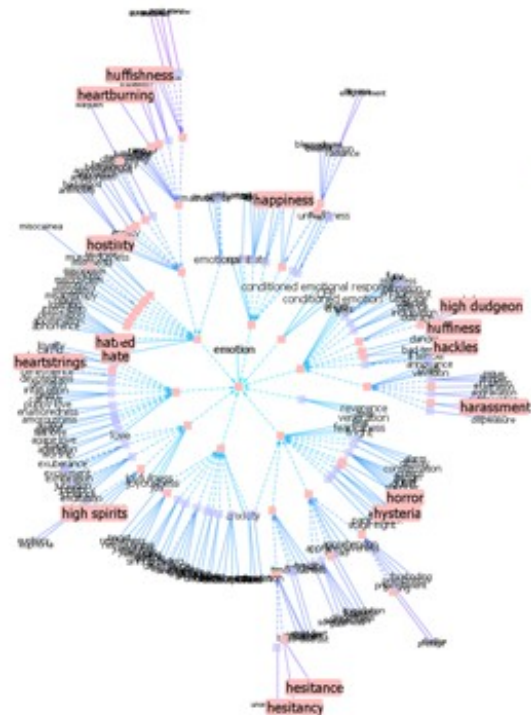
## Procesamiento básico

WordNet es una enorme red de palabras

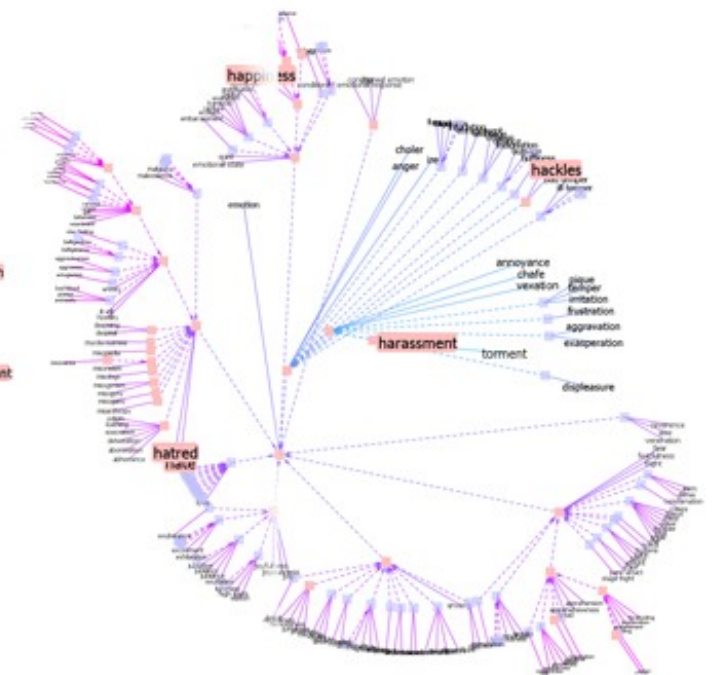
- 155287 palabras organizadas en 117659 synsets



→  
Zoom out



→  
Zoom in



procesamiento de lenguaje natural

Spanish

Traduzca al...

🔍

bn:00045156n | Sustantivo | Concepto | Categorías: Procesamiento de le...

- ES procesamiento de lenguajes naturales
- Procesamiento del lenguaje natural
- procesamiento de lenguaje natural • NLP
- Procesamiento de lenguas naturales



See more

El procesamiento de lenguaje natural o de lenguas naturales,[1][2] abreviado PLN[3][4], es un campo de las ciencias de la computación, de la inteligencia artificial y de la lingüística que estudia las interacciones entre las computadoras y el lenguaje humano, así como los detalles computacionales de las lenguas naturales.

English: computer science



DEFINICIONES

RELACIONES

FUENTES

Spanish > Más idiomas...

IS A	informática • especialidad • especialidad • campo de estudio • disciplina académica +4 relations
HAS PART	lingüística computacional • procesamiento de lenguajes naturales • word embedding
PART OF	procesamiento de lenguajes naturales
HAS KIND	EN Morphological parsing • codificación de caracteres chinos • reconocimiento del habla • generación de lenguajes naturales • segmentación del texto +6 relations
FACET OF	Extracción de la información
STUDIES	correferencia • semántica léxica • traducción automática • EN relationship extraction • reconocimiento de nombres de entidades +19 relations

## - VECTORIZACIÓN DE DOCUMENTOS Y RANKING -

## Matriz términos-documentos

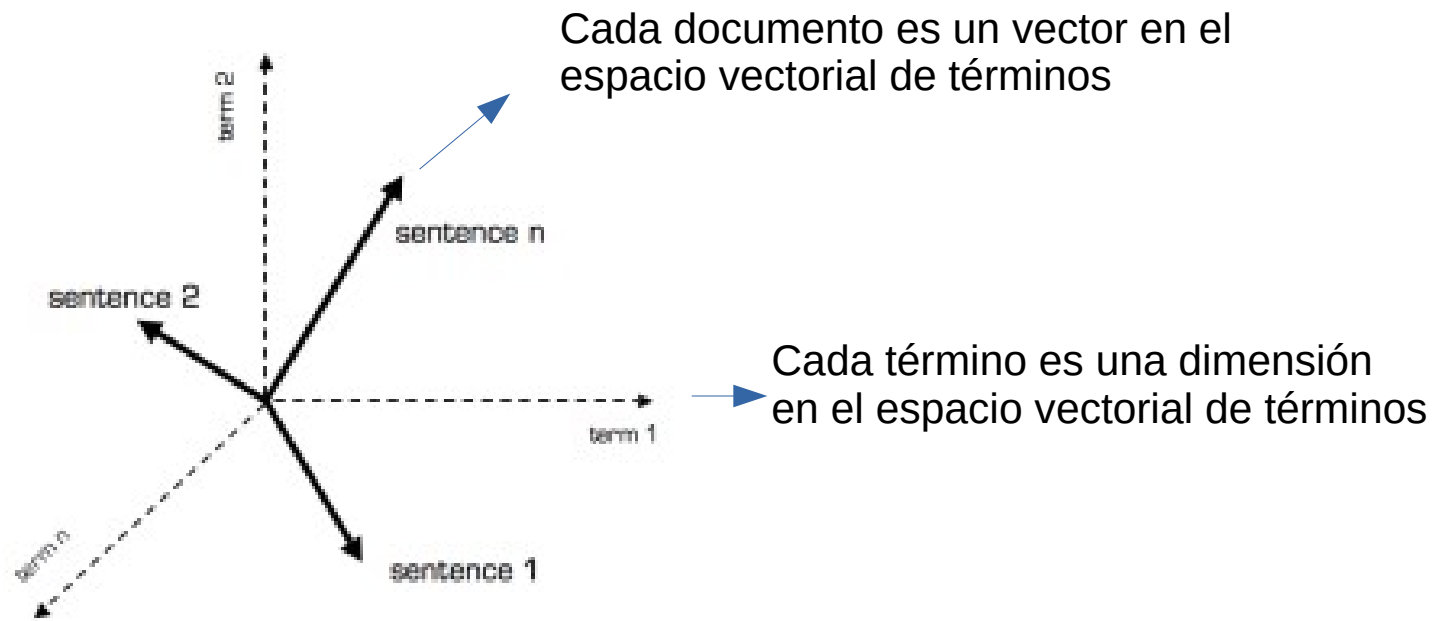
documentos →

	Antonio y Cleopatra	Julio Cesar	La Tempestad	Hamlet	Otelo	Macbeth	...
Antonio	157	73	0	0	0	1	
Brutus	4	157	0	2	0	0	
Cesar	232	227	0	2	1	0	
Calpurnia	0	10	0	0	0	0	
Cleopatra	57	0	0	0	0	0	
...		...					

↑  
términos

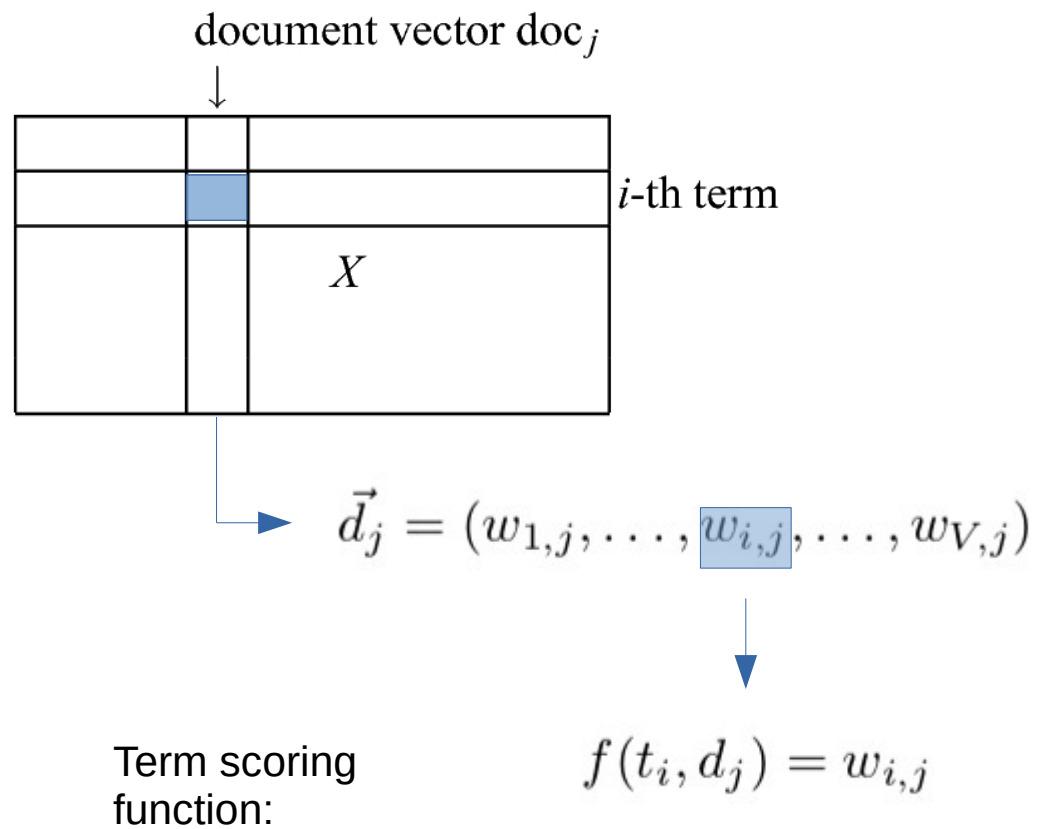
→ vectorización

## Vector-space model





## Vector-space model



## BM25



Prueba y error (intento 25)

$f_{i,j}$  : # occs. de  $t_i$  en  $d_j$

$N$  : # docs

$n_i$  : # docs donde  $t_i$  ocurre

$l(d_j)$  : # tokens en  $d_j$

$l_{avg}$  : largo promedio

$$w_{i,j} = \frac{f_{i,j} \cdot (k_1 + 1)}{k_1 \cdot \left[ (1 - b) + b \cdot \frac{l(d_j)}{l_{avg}} \right] + f_{i,j}} \cdot \log \left( \frac{N}{n_i} \right)$$

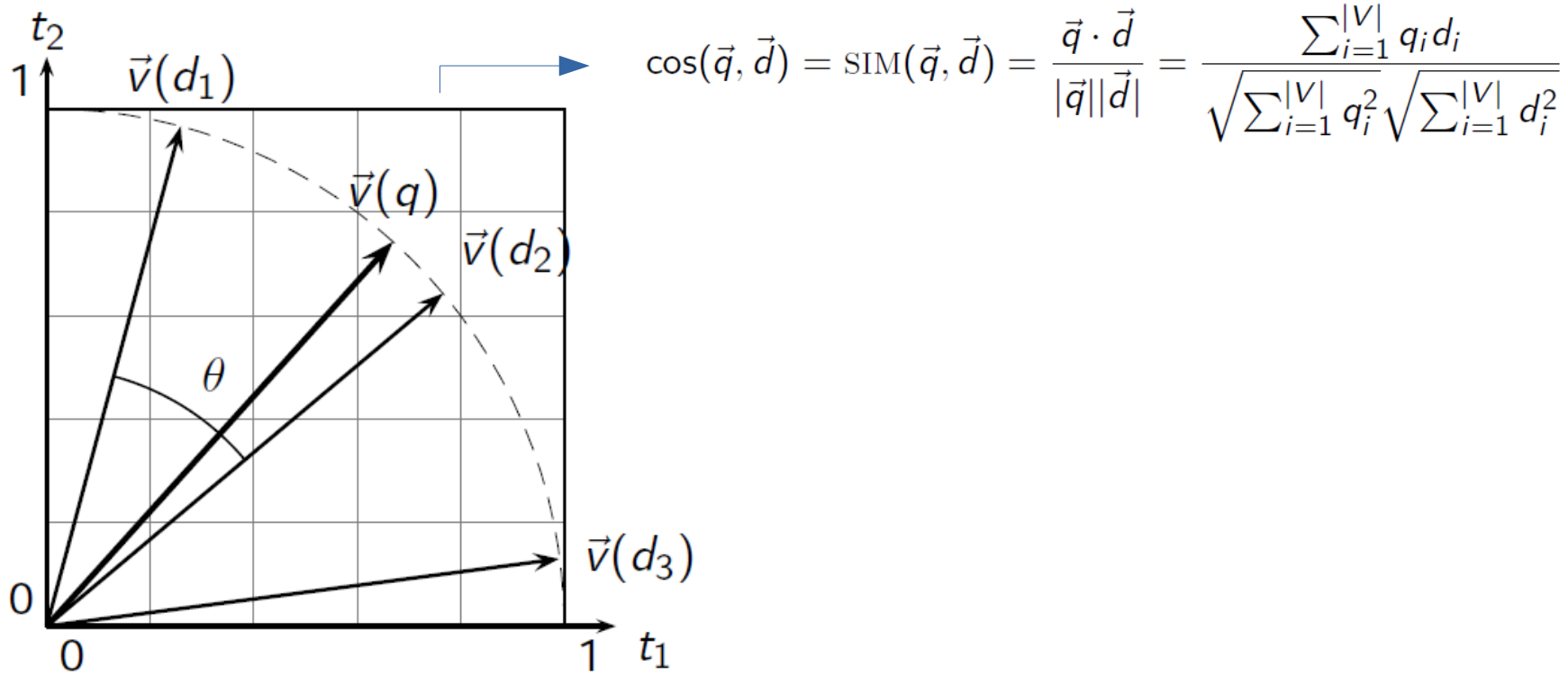
$$b \in [0, 1], \quad k_1 > 0$$

Empírico:  $b \approx 0.75$

$k_1 \approx 1.2$

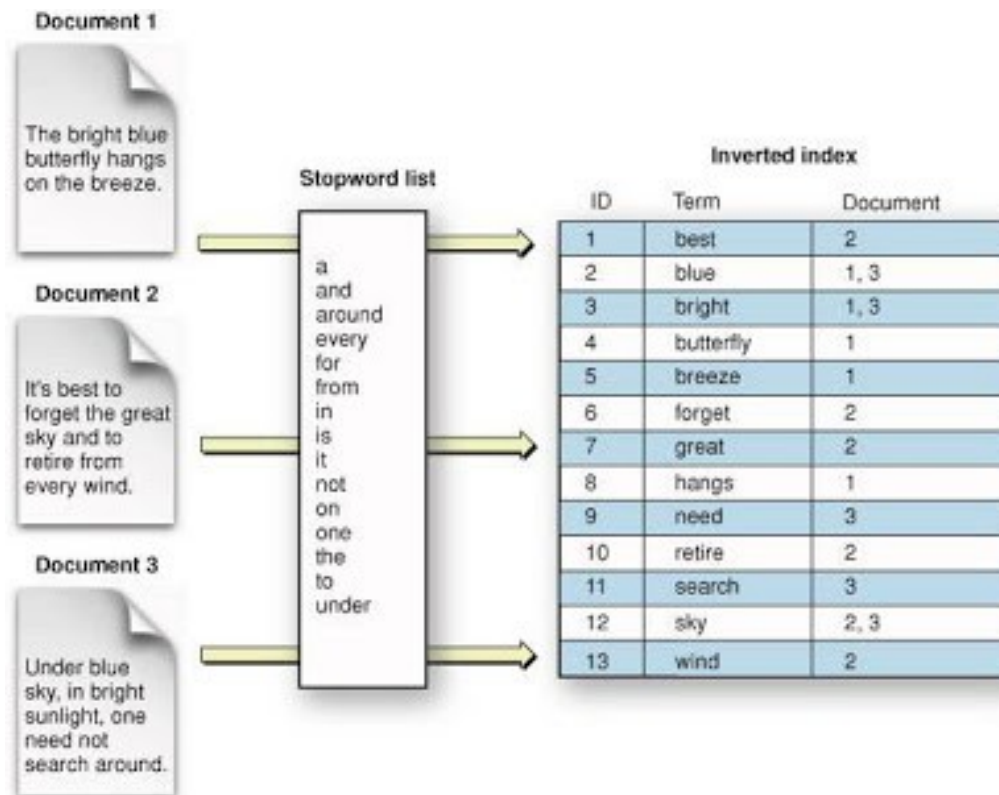
## Document ranking

Funciones de proximidad entre vectores:

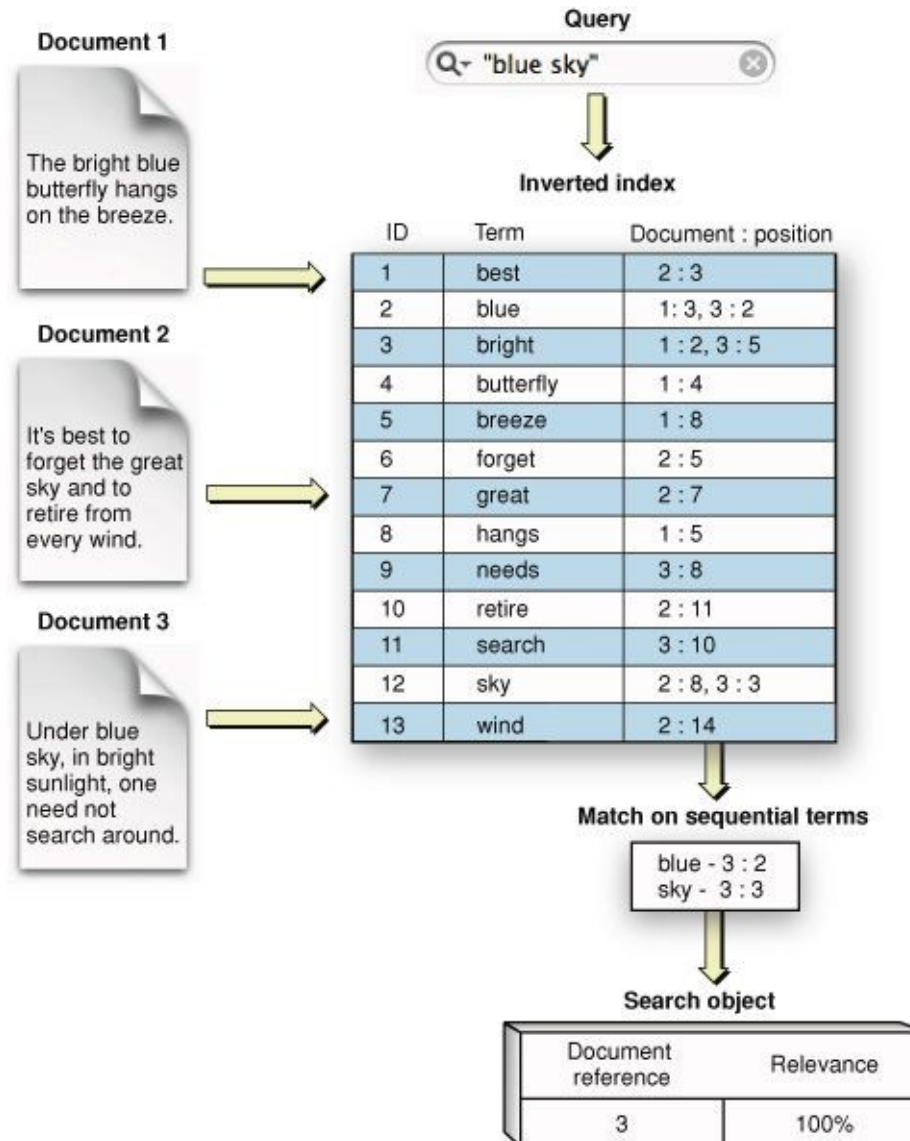


BM25 está basado en esta idea.

## Índice invertido:

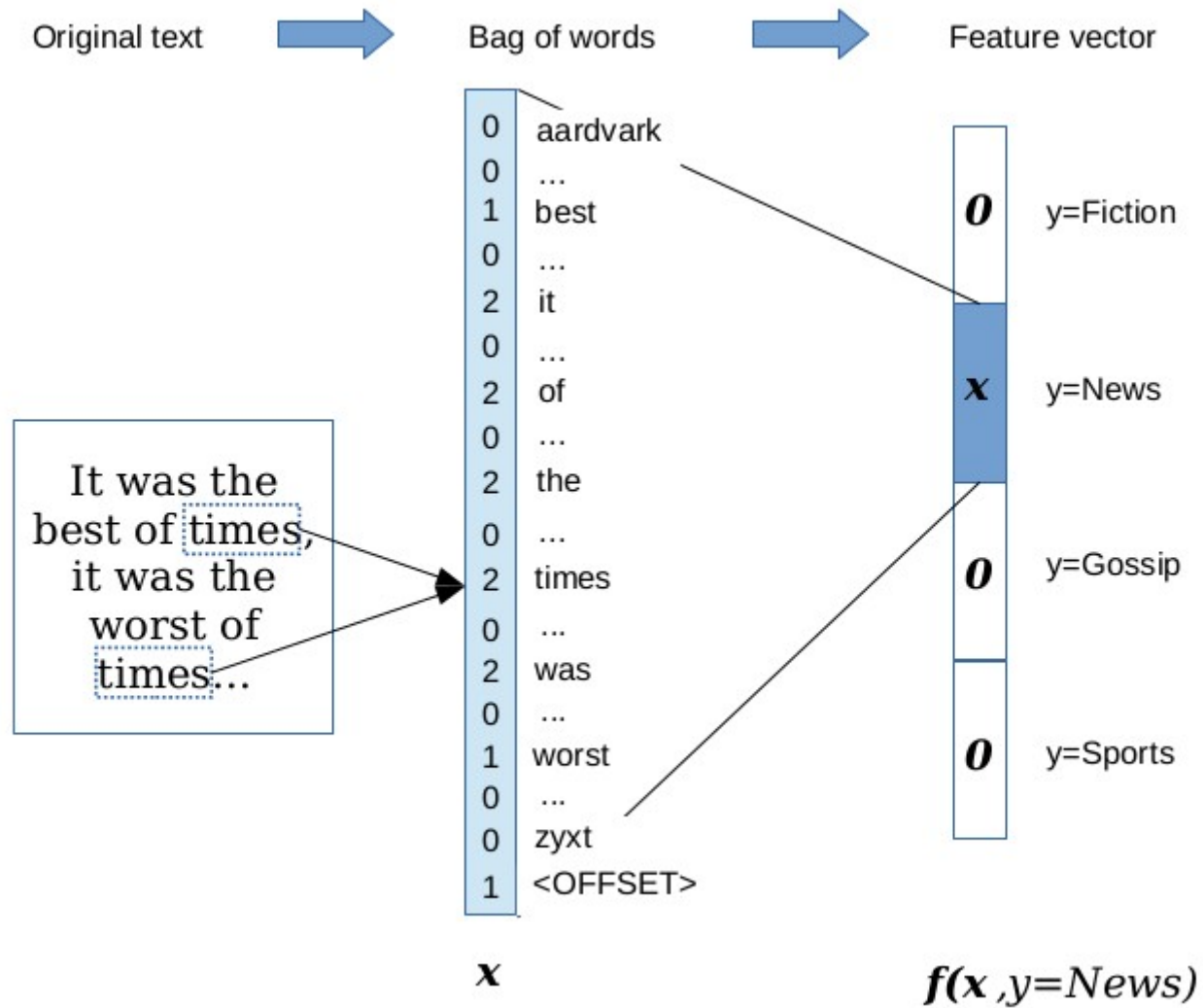


## Ranking:



## - CLASIFICACIÓN DE DOCUMENTOS -

## BOW



$f_{i,j}$  : # occs. de ti en dj

$\max f_{l,j}$

Term scoring functions:

$N$  : # docs

$n_i$  : # docs donde ti ocurre

- Tf: 
$$Tf_{i,j} = \frac{f_{i,j}}{\max f_{l,j}}$$

- Tf corregido: 
$$w_{i,j} = \begin{cases} 1 + \log_{10} f_{i,j} & \text{if } f_{i,j} > 0 \\ 0 & \text{e.t.o.c.} \end{cases}$$

- Idf: 
$$\text{idf}_{ti} = \log_{10} \frac{N}{n_i}$$

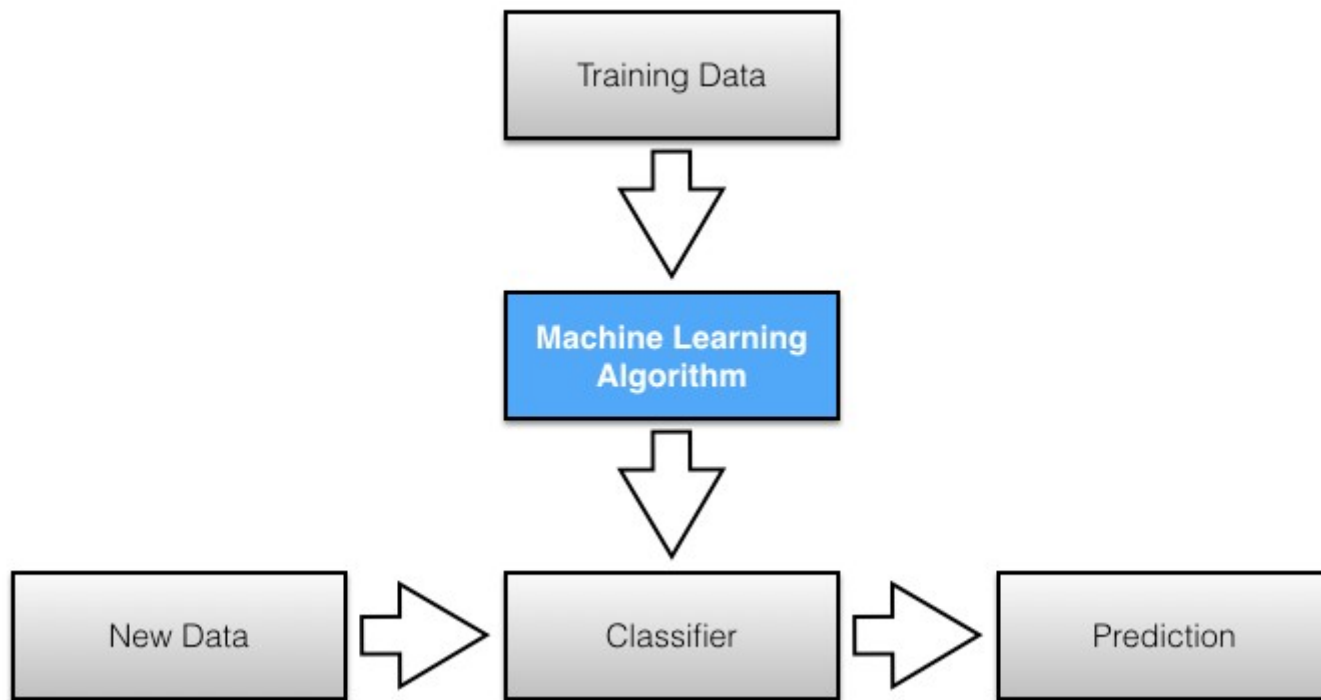
- Tf-Idf (Salton): 
$$w_{i,j} = (1 + \log f_{l,j}) \cdot \log \frac{N}{n_i}$$

- Tf-Idf: 
$$w_{i,j} = \frac{f_{i,j}}{\max f_{l,j}} \cdot \log \frac{N}{n_i}$$



## Clasificación de documentos

Síntesis. El enfoque de ML (clásico)



- Anexos -

## Preprocesamiento de texto en NLTK

### Procesamiento básico Web:

---

```
> import nltk
> from urllib import urlopen
> url = "http://www.gutenberg.org/files/2554/2554.txt"
> raw = urlopen(url).read()
```

---

### Tokenización y creación del objeto texto:

---

```
> tokens = nltk.word_tokenize(raw)
> text = nltk.Text(tokens)
```

---

### Ahora podemos hacer NLP sobre el texto:

---

```
> text.collocations()
> ...
```

---

## Preprocesamiento de texto en NLTK

### Procesamiento de HTML:

---

```
> url = "http://nltk.org"  
> html = urlopen(url).read()  
> raw = nltk.clean_html(html)
```

---

### Repetimos el pipe anterior:

---

```
> tokens = nltk.word_tokenize(raw)  
> text = nltk.Text(tokens)  
> text.collocations()
```

---

### Construir el vocabulario (minúsculas y sorted set):

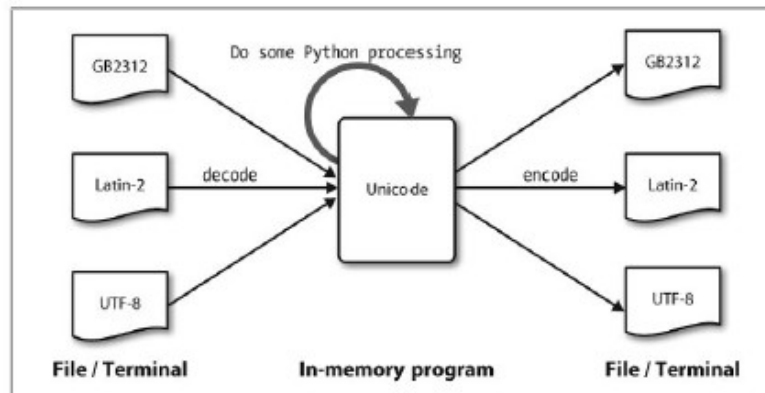
---

```
> words = [w.lower() for w in text]  
> vocab = sorted(set(words))
```

---

## Preprocesamiento de texto en NLTK

Leer con decode, procesar en Unicode, print con encode (render glyphs).



### Procesamiento de Unicode (Spanish!):

---

```
> url = "http://www.inf.utfsm.cl"
> html = urlopen(url).read()
> raw = nltk.clean_html(html)
> decoded = raw.decode('utf8')
> print decoded.encode('latin2')
```

---

## Preprocesamiento de texto en NLTK

### Stemmers:

---

```
> porter = nltk.PorterStemmer()
> lancaster = nltk.LancasterStemmer()
> [porter.stem(t) for t in tokens]
> [lancaster.stem(t) for t in tokens]
```

---

### Lematizador (stemmer + corpus checking):

---

```
> wnl = nltk.WordNetLemmatizer()
> [wnl.lemmatize(t) for t in tokens]
```

---

## Preprocesamiento de texto en NLTK

Segmentador para texto raw en inglés:

---

```
> url = "http://www.gutenberg.org/files/2554/2554.txt"
> raw = urlopen(url).read()
> sent_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
> sents = sent_tokenizer.tokenize(raw)
```

---

Entrega una lista de sentencias:

---

```
> len(sents)
> print sents[1].encode('latin2')
```

---

## Preprocesamiento de texto en NLTK

Podemos mejorar el tokenizer de NLTK, agregando expresiones regulares que queremos detectar como unigramas:

---

```
> pattern = r'''(?x)
...      ([A-Z]\.)+          # abreviaciones (U.S.A.)
...      | \w+(-\w+)*        # palabras con guiones
...      | \$?\d+(\.\d+)     # precios
...      | \.\.\.           # elipsis
...      | [][.,;"'()? :-_]  # tokenizadores
...      '''
>>> nltk.regexp_tokenize(text,pattern)
```

---