IIC3670 Procesamiento de Lenguaje Natural

https://github.com/marcelomendoza/IIC3670

- OUTLINE -

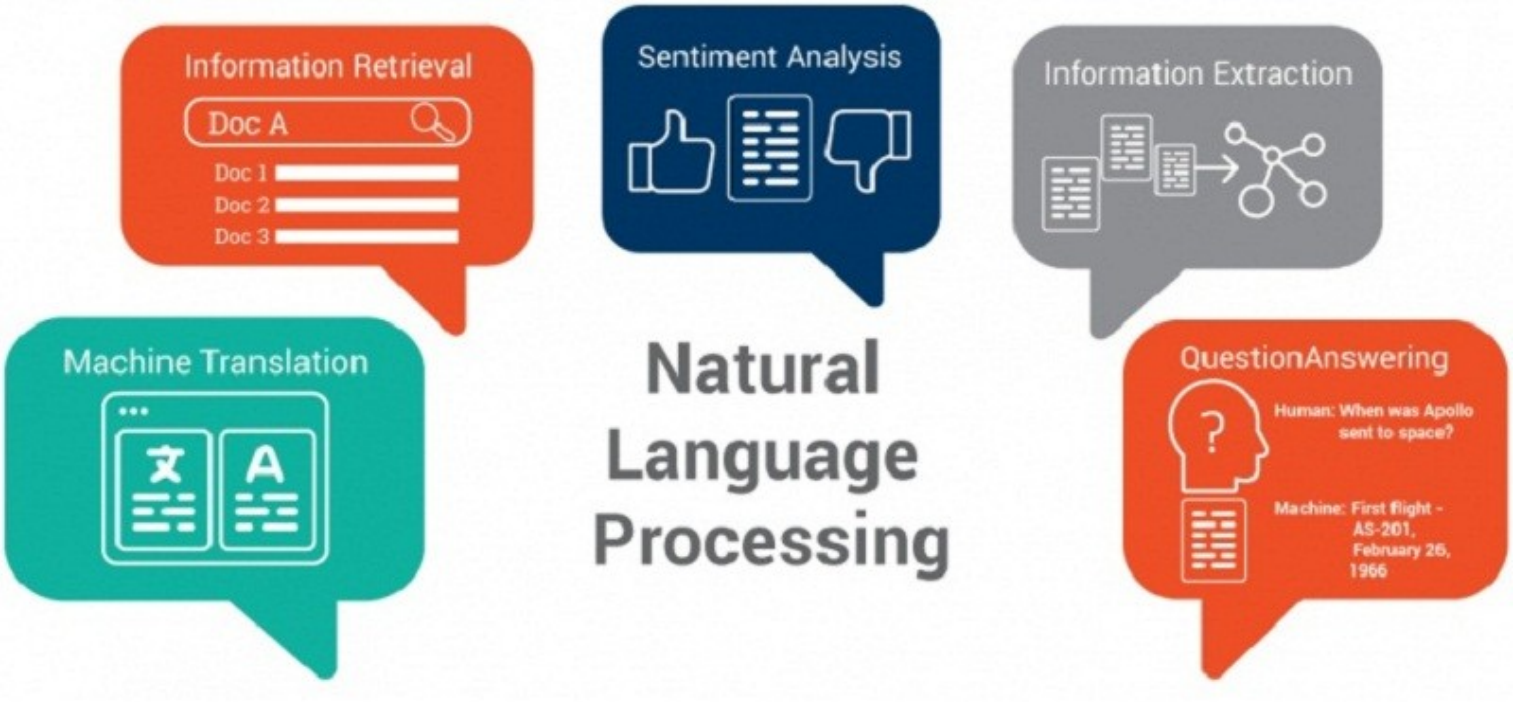
¿Qué vamos a ver?

- Ranking y clasificación de documentos
- Word2vec, Glove, FastText, Flair
- Transformer, BERT, RoBERTa
- GPT, ChatGPT, alignment
- COT, halucinaciones, fairness

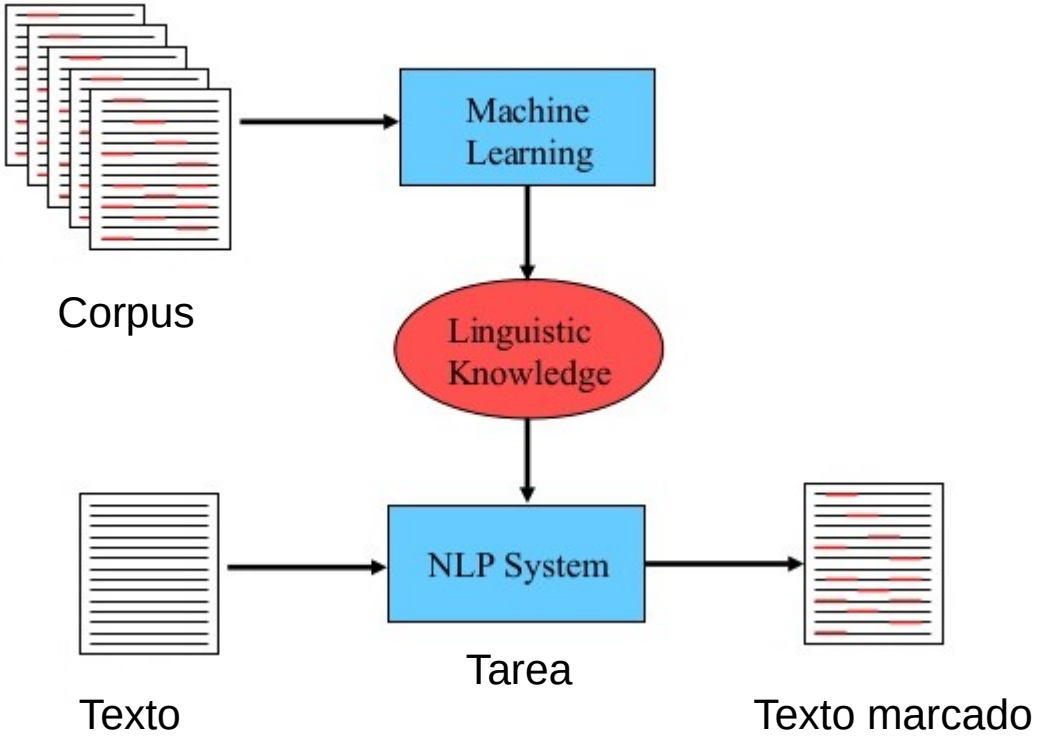
- INTRODUCCIÓN -

Introducción

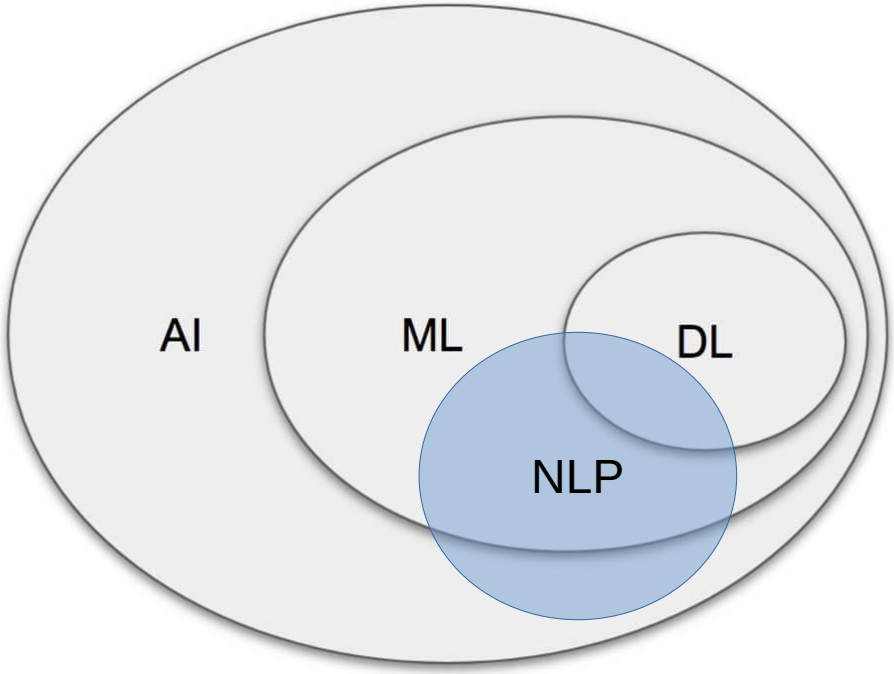
Síntesis. ¿Cuáles tareas aborda NLP?



El enfoque de NLP (clásico)

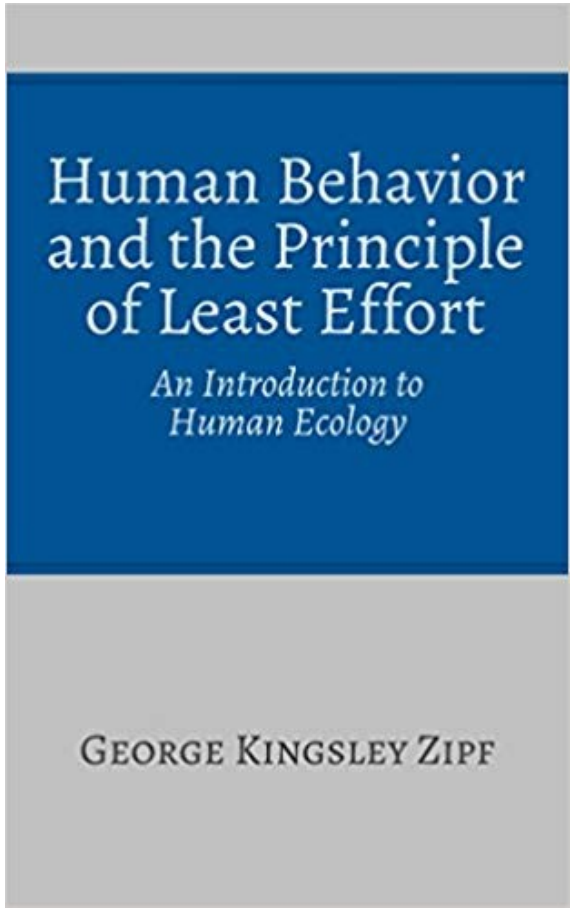


¿Dónde se ubica NLP?



- LEYES DEL TEXTO Y PROCESAMIENTO BÁSICO -

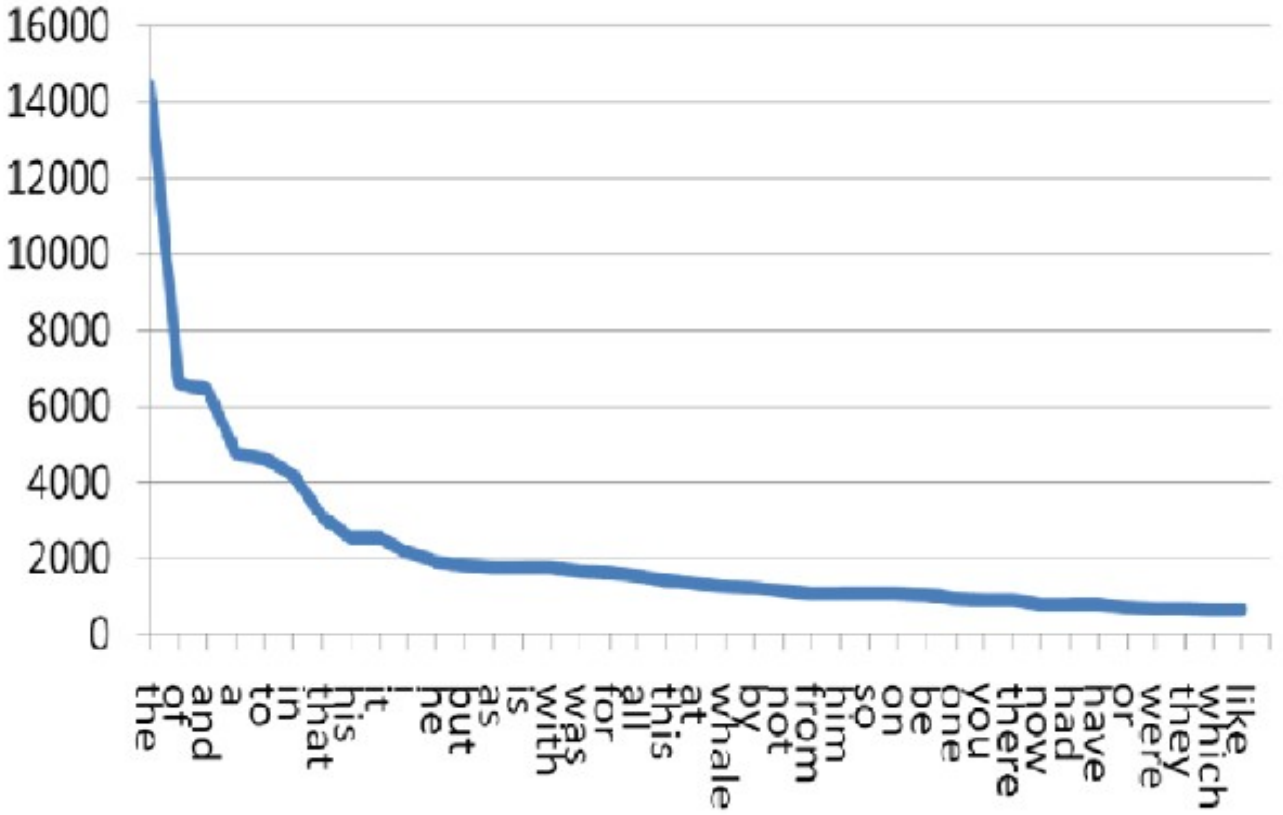
Leyes del texto



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

Leyes del texto

Zipf para Reuters[2]



[2]Dataset de noticias, disponible on-line

Leyes del texto

θ : pendiente de la curva log-log

n : # tokens

r : ranking de la palabra

f : # ocurrencias de la palabra

Ley de Zipf: $f \sim \frac{1}{r}$

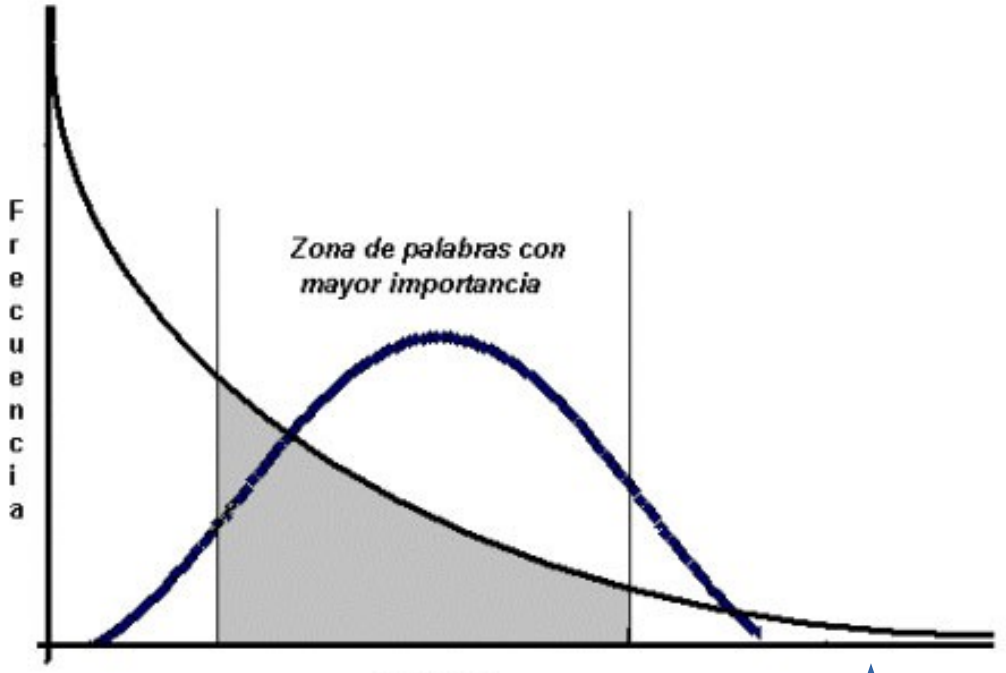
$$f \sim \frac{1}{r^\theta}$$

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

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

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

Leyes del texto

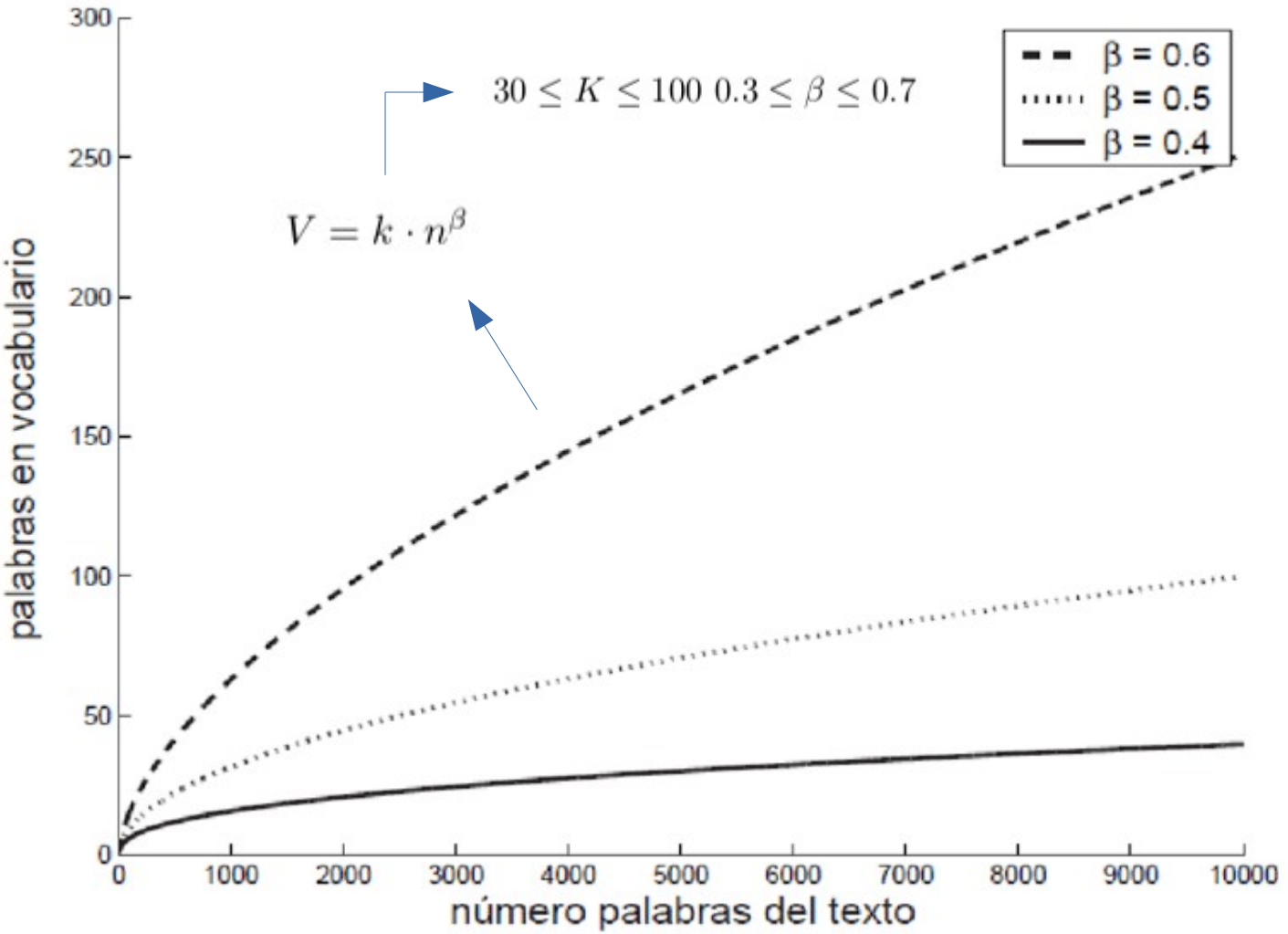


stopwords

keywords

outliers (e.g. typos)

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.

Procesamiento básico

Lematización

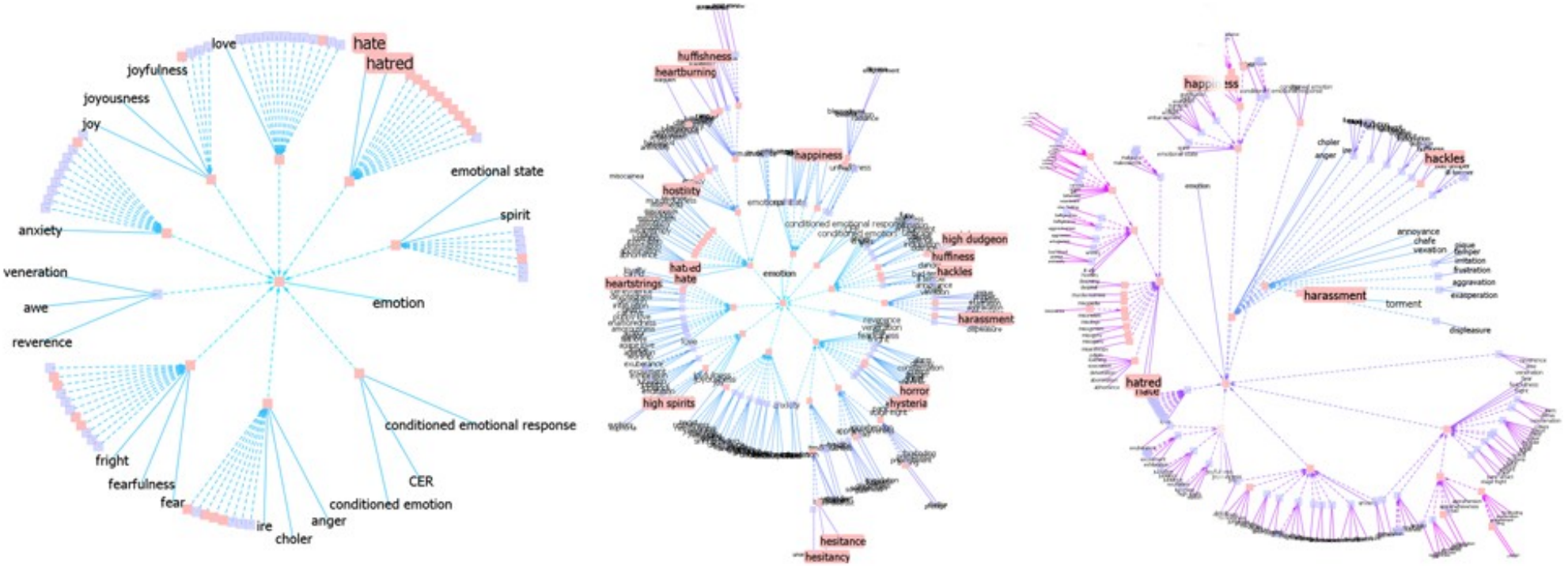
WordNet lemmatizer

- Reducir formas inflecciones 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*)

Procesamiento básico

WordNet es una enorme red de palabras

- 155287 palabras organizadas en 117659 synsets



Zoom out → Zoom in

- VECTORIZACIÓN DE DOCUMENTOS Y RANKING -

Matriz términos-documentos

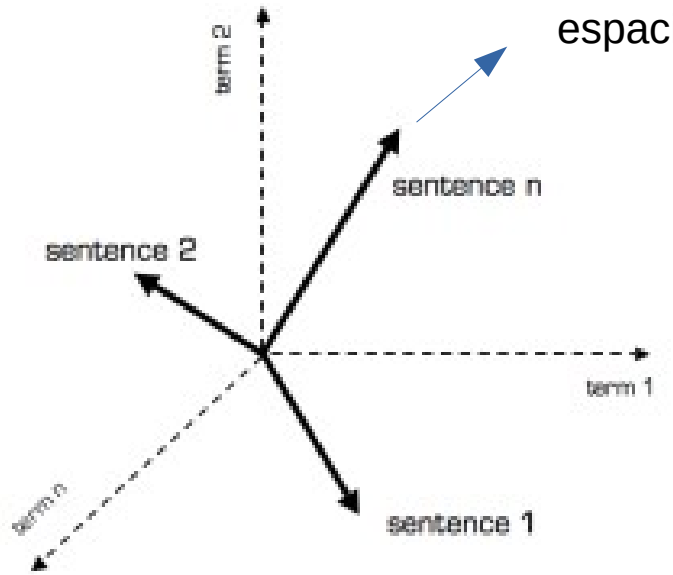
documentos →

	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	
...		...					

vectorización

términos

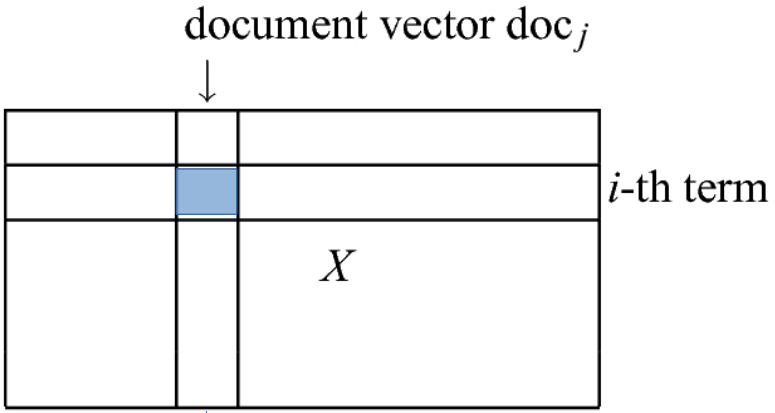
Vector-space model



Cada documento es un vector en el espacio vectorial de términos

Cada término es una dimensión en el espacio vectorial de términos

Vector-space model



$$\vec{d_j} = (w_{1,j}, \ldots, w_{i,j}, \ldots, w_{V,j})$$

Term scoring function:

$$f(t_i, d_j) = w_{i,j}$$

BM25

Prueba y error (intento 25)

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

N : # docs

n_i : # docs donde ti ocurre

$l(d_j)$: # tokens en dj

l_{avg} : largo promedio

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

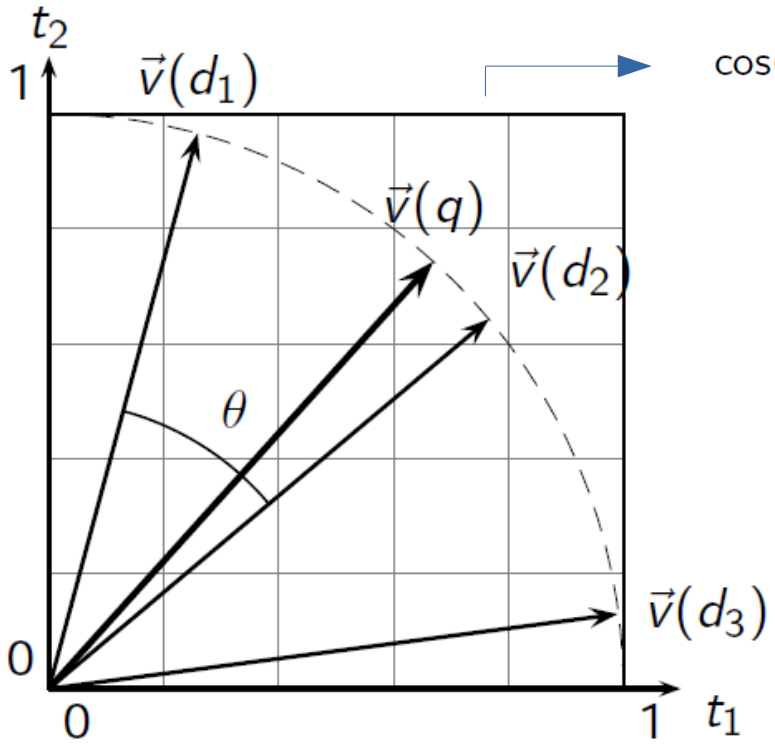
$$b \in [0, 1], \quad k_1 > 0$$

Empírico: $b \approx 0.75$

$k_1 \approx 1.2$

Document ranking

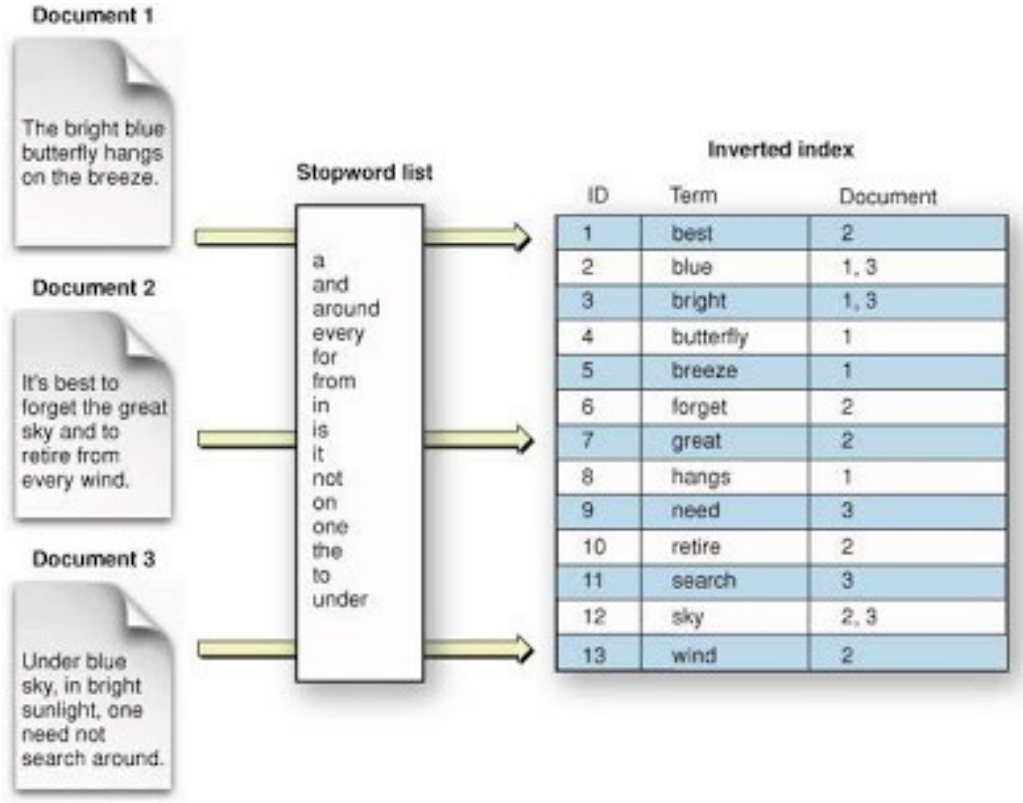
Funciones de proximidad entre vectores:



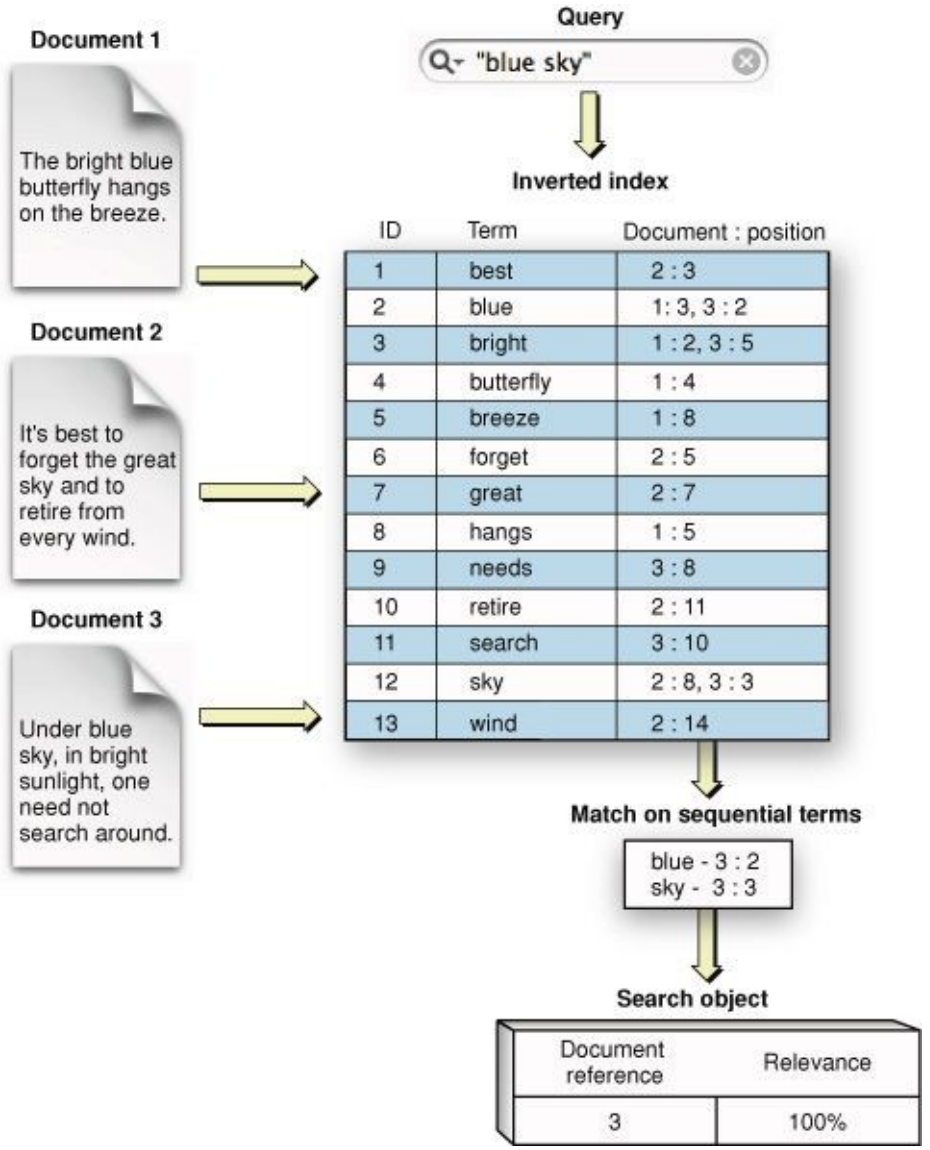
$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

BM25 está basado en esta idea.

Índice invertido:

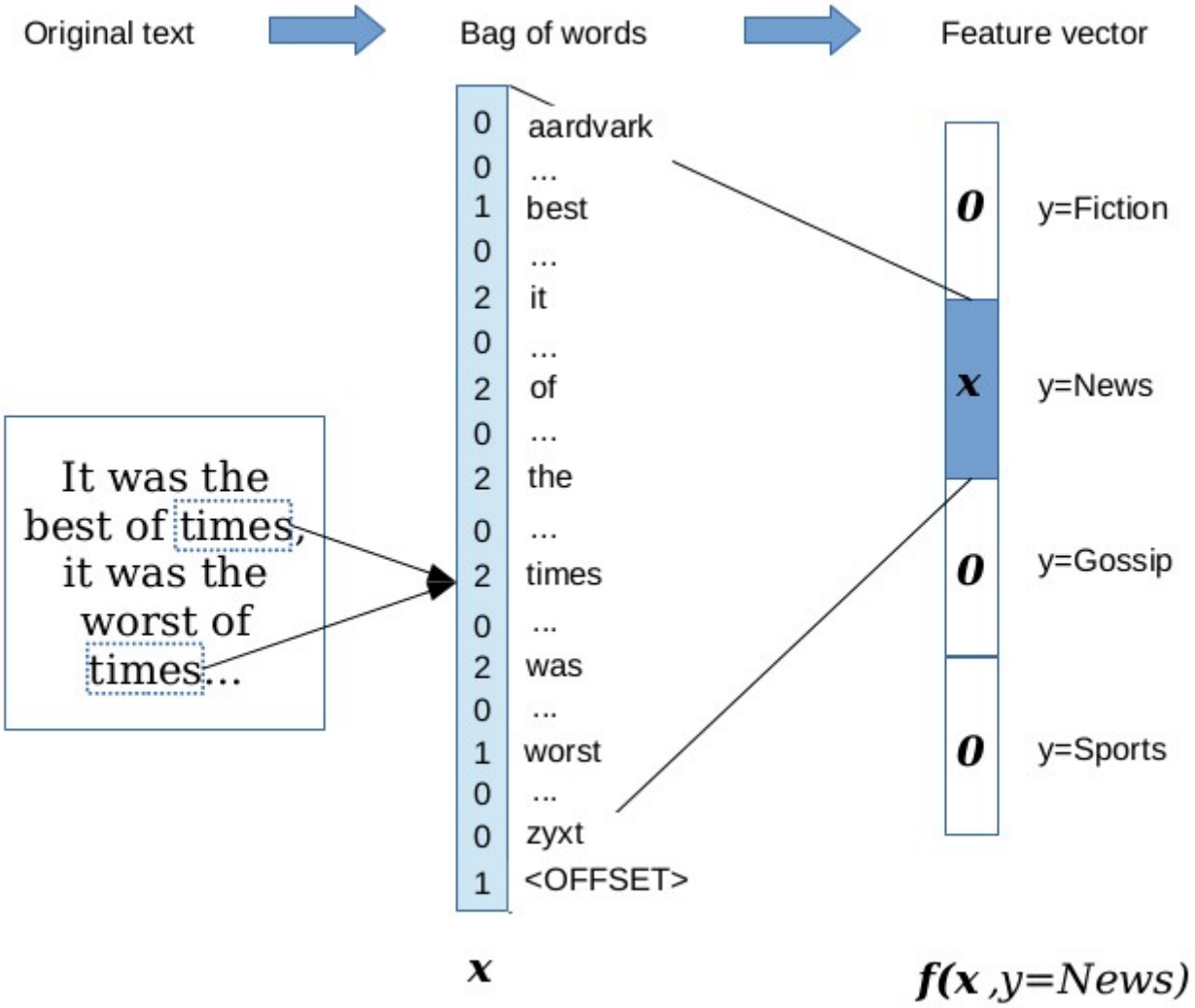


Ranking:


Query
"blue sky"
Inverted index
Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
ID
Term
Document : position
1 best 2 : 3
2 blue 1: 3, 3 : 2
3 bright 1 : 2, 3 : 5
4 butterfly 1 : 4
5 breeze 1 : 8
6 forget 2 : 5
7 great 2 : 7
8 hangs 1 : 5
9 needs 3 : 8
10 retire 2 : 11
11 search 3 : 10
12 sky 2 : 8, 3 : 3
13 wind 2 : 14
Match on sequential terms
blue - 3 : 2
sky - 3 : 3
Search object
Document reference
Relevance
3
100%

- CLASIFICACIÓN DE DOCUMENTOS -

BOW


Original text
Bag of words
Feature vector
It was the best of times, it was the worst of times...
0 aardvark
0 ...
1 best
0 ...
2 it
0 ...
2 of
0 ...
2 the
0 ...
2 times
0 ...
2 was
0 ...
1 worst
0 ...
0 zyxt
1 <OFFSET>
0 y=Fiction
x y=News
0 y=Gossip
0 y=Sports
x
f(x ,y=News)

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

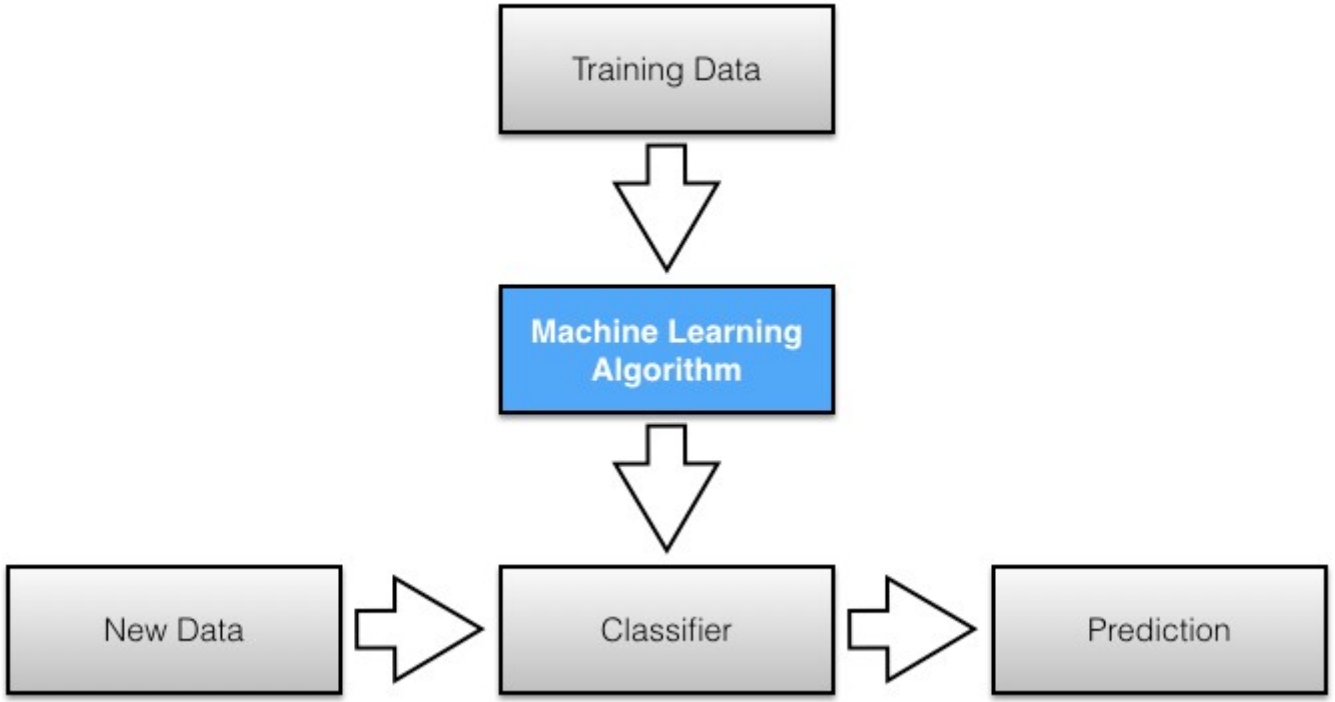
- Idf: $$\text{idf}_{ti} = \log_{10} \frac{N}{n_i}$$

- 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)
```

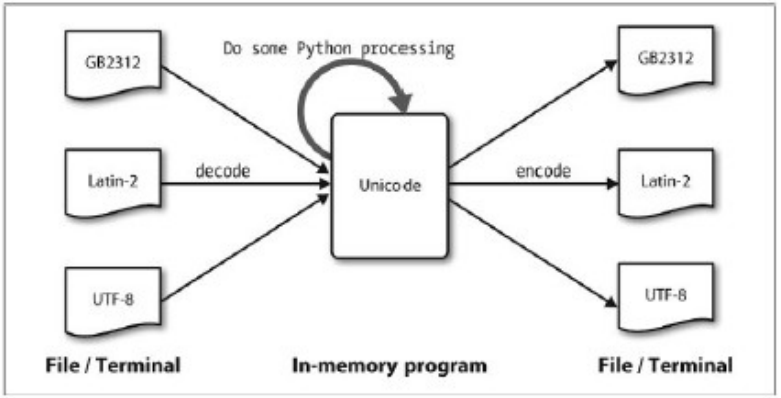
Repetimos el pipe anterior:

```
> 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))
```

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(hmtl)
> 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)
```

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)
```