# Notas TFM

# Tokenización

oracion = "En 1884, con treinta y dos años, Santiago Ramón y Cajal se trasladó a Valencia a ocupar su cátedra."  
oracion.split()

['En',  
 '1884,',  
 'con',  
 'treinta',  
 'y',  
 'dos',  
 'años,',  
 'Santiago',  
 'Ramón',  
 'y',  
 'Cajal',  
 'se',  
 'trasladó',  
 'a',  
 'Valencia',  
 'a',  
 'ocupar',  
 'su',  
 'cátedra.']

import numpy as np  
import pandas as pd  
  
token\_secuencia = str.split(oracion)  
vocab = sorted(set(token\_secuencia))  
', '.join(vocab)

'1884,, Cajal, En, Ramón, Santiago, Valencia, a, años,, con, cátedra., dos, ocupar, se, su, trasladó, treinta, y'

num\_tokens = len(token\_secuencia)  
tam\_vocab = len(vocab)  
vectores\_onehot = np.zeros((num\_tokens, tam\_vocab), int)  
for i, plb in enumerate(token\_secuencia):  
 vectores\_onehot[i, vocab.index(plb)] = 1  
' '.join(vocab)

'1884, Cajal En Ramón Santiago Valencia a años, con cátedra. dos ocupar se su trasladó treinta y'

df = pd.DataFrame(vectores\_onehot, columns=vocab)  
df

1884,

Cajal

En

Ramón

Santiago

Valencia

a

años,

con

cátedra.

dos

ocupar

se

su

trasladó

treinta

y

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

2

0

0

0

0

0

0

0

0

1

0

0

0

0

0

0

0

0

3

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

0

4

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

5

0

0

0

0

0

0

0

0

0

0

1

0

0

0

0

0

0

6

0

0

0

0

0

0

0

1

0

0

0

0

0

0

0

0

0

7

0

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

0

8

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

9

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

10

0

1

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

11

0

0

0

0

0

0

0

0

0

0

0

0

1

0

0

0

0

12

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

0

0

13

0

0

0

0

0

0

1

0

0

0

0

0

0

0

0

0

0

14

0

0

0

0

0

1

0

0

0

0

0

0

0

0

0

0

0

15

0

0

0

0

0

0

1

0

0

0

0

0

0

0

0

0

0

16

0

0

0

0

0

0

0

0

0

0

0

1

0

0

0

0

0

17

0

0

0

0

0

0

0

0

0

0

0

0

0

1

0

0

0

18

0

0

0

0

0

0

0

0

0

1

0

0

0

0

0

0

0

En esta presentación df muestra un documento de una sola oración en la que cada fila es un vector de una sola palabra. A esto corresponde un vector *onehot*: 1 significa que la palabra está activada, 0 que no.

Mediante este tipo de tablas es difícil que se pierda mucha información.

Este es el punto de partida de métodos como redes neuronales, modelos lingüísticos de secuencia a secuencia y generadores.

## Bolsas de palabras

Para tener un control sobre la frecuencia con la que aparece cada palabra en un documento se crea un saco de palabras ( *bag of words* ).

sdp\_oracion = {}  
for token in oracion.split():  
 sdp\_oracion[token] = 1  
sorted(sdp\_oracion.items())

[(‘1884,’, 1), (‘Cajal’, 1), (‘En’, 1), (‘Ramón’, 1), (‘Santiago’, 1), (‘Valencia’, 1), (‘a’, 1), (‘años,’, 1), (‘con’, 1), (‘cátedra.’, 1), (‘dos’, 1), (‘ocupar’, 1), (‘se’, 1), (‘su’, 1), (‘trasladó’, 1), (‘treinta’, 1), (‘y’, 1)]

Bajo este principio se pueden agregar más oraciones para hacer más grande el documento.

oraciones = """En 1884, con treinta y dos años, Santiago Ramón y Cajal se trasladó a Valencia a ocupar su cátedra.\n"""  
oraciones += """Llegó en enero y, junto a su familia, se hospedó provisionalmente en una fonda situada en la plaza del Mercado, cerca de la vieja Lonja de la Seda.\n"""  
oraciones += """Pronto encontró una casita en la calle dude las Avellanas, donde pocos días después nacía su hija Paula.\n"""  
oraciones += """Ahora tenía tres: los dos mayores eran una muchacha, Fe, y un chico, Santiago."""  
corpus = {}  
for i, orac in enumerate(oraciones.split('\n')):  
 corpus['orac{}'.format(i)] = dict((tok, 1) for tok in orac.split())  
df = pd.DataFrame.from\_records(corpus).fillna(0).astype(int).T  
df

En

1884,

con

treinta

y

dos

años,

Santiago

Ramón

Cajal

…

tenía

tres:

los

mayores

eran

muchacha,

Fe,

un

chico,

Santiago.

orac0

1

1

1

1

1

1

1

1

1

1

…

0

0

0

0

0

0

0

0

0

0

orac1

0

0

0

0

0

0

0

0

0

0

…

0

0

0

0

0

0

0

0

0

0

orac2

0

0

0

0

0

0

0

0

0

0

…

0

0

0

0

0

0

0

0

0

0

orac3

0

0

0

0

1

1

0

0

0

0

…

1

1

1

1

1

1

1

1

1

1

4 rows × 62 columns

## Producto escalar

Transformando documentos en tablas se pueden realizar productos escalares.

Mediante el método .dot es posible saber cuántas palabras se traslapan entre las distintas oraciones. Se podría decir que esta es una medida de similaridad. En este caso, se contrasta contra la primera oración orac0.

df = df.T  
df.orac0.dot(df.orac1)

3

df.orac0.dot(df.orac2)

1

df.orac0.dot(df.orac3)

2

Sabemos que tanto orac0 como orac3 tienen a la palabra Santiago, pero el código que hemos puesto hasta ahora no lo detecta.

[(k, v) for (k, v) in (df.orac0 & df.orac3).items() if v]

[(‘y’, 1), (‘dos’, 1)]

Esto sucede porque sin una orden explícita el programa considera como palabras distintas a Santiago y Santiago.. Para solucionar este problema, se utiliza una técnica llamada **expresiones regulares** ( *regular expresions* o *regex* ):

import re  
patron = re.compile(r"([-\s.,;!?])+")  
tokens = patron.split(oracion)  
tokens = [x for x in tokens if x and x not in '- \t\n.,;!?']  
tokens

[‘En’, ‘1884’, ‘con’, ‘treinta’, ‘y’, ‘dos’, ‘años’, ‘Santiago’, ‘Ramón’, ‘y’, ‘Cajal’, ‘se’, ‘trasladó’, ‘a’, ‘Valencia’, ‘a’, ‘ocupar’, ‘su’, ‘cátedra’]

Existen varios módulos de Python que ayudan a realizar esta tarea, como lo son:

* NLTK: más popular para procesamiento de lenguaje natural.
* spaCy: preciso, flexible, rápido y nativo de Python.
* Stanford CoreNLP: más preciso, menos flexible, rápido y basado en Java 8.

Para este ejemplo, utilizaremos TreebankWordTokenizer para realizar las filtraciones sin que tengamos preocuparnos por ellas. Además, esta función permite conservar signos de puntuación que podrían ser relevantes en algún momento.

from nltk.tokenize import TreebankWordTokenizer  
tokenizador = TreebankWordTokenizer()  
tokenizador.tokenize(oracion)

[‘En’, ‘1884’, ‘,’, ‘con’, ‘treinta’, ‘y’, ‘dos’, ‘años’, ‘,’, ‘Santiago’, ‘Ramón’, ‘y’, ‘Cajal’, ‘se’, ‘trasladó’, ‘a’, ‘Valencia’, ‘a’, ‘ocupar’, ‘su’, ‘cátedra’, ‘.’]

## N-grams

Detectar una palabra muchas veces no es suficiente porque el sentido de ese token depende de sus vecinos, como lo es “Santiago Ramón y Cajal”. Para abordar esta situación, se determinan los *n\_grams*, cuya *n* corresponde al número de vecinos contiguos que se extraen de las oraciones. La tokenización que hicimos previamente utiliza 1-gram. Para el nombre propio de Cajal, utilizaríamos 4-gram, como se ejemplifica a continuación:

from nltk.util import ngrams  
  
tres\_grams = list(ngrams(tokens, 3))  
[" ".join(x) for x in tres\_grams]

[‘En 1884 con’, ‘1884 con treinta’, ‘con treinta y’, ‘treinta y dos’, ‘y dos años’, ‘dos años Santiago’, ‘años Santiago Ramón’, ‘Santiago Ramón y’, ‘Ramón y Cajal’, ‘y Cajal se’, ‘Cajal se trasladó’, ‘se trasladó a’, ‘trasladó a Valencia’, ‘a Valencia a’, ‘Valencia a ocupar’, ‘a ocupar su’, ‘ocupar su cátedra’]

cuatro\_grams = list(ngrams(tokens, 4))  
[" ".join(x) for x in cuatro\_grams]

[‘En 1884 con treinta’, ‘1884 con treinta y’, ‘con treinta y dos’, ‘treinta y dos años’, ‘y dos años Santiago’, ‘dos años Santiago Ramón’, ‘años Santiago Ramón y’, ‘Santiago Ramón y Cajal’, ‘Ramón y Cajal se’, ‘y Cajal se trasladó’, ‘Cajal se trasladó a’, ‘se trasladó a Valencia’, ‘trasladó a Valencia a’, ‘a Valencia a ocupar’, ‘Valencia a ocupar su’, ‘a ocupar su cátedra’]

El problema de esta aproximación es que en un documento, casos como “Santiago Ramón y Cajal” o “treinta y dos” ocurrirán rara vez. Eso implica que difícilmente existirá una correlación con otras palabras que permita identificar un tema contenido en documentos. Adicionalmente, cada *n-gram* aumenta exponencialmente el tamaño del documento, por lo que no es viable.

Por otra parte, también se encuentran palabras que no aportan mayor información, como son los artículos y preposiciones. llamadas “palabras vacías” ( *stop\_words* ). Existen razones computacionales para quitarlas, aunque presentan sus propios inconvenientes.

from nltk.corpus import stopwords  
  
pal\_vacias = stopwords.words('spanish')  
pal\_vacias[:7]

[‘de’, ‘la’, ‘que’, ‘el’, ‘en’, ‘y’, ‘a’]

tokens = ['el', 'niño', 'salió', 'a', 'jugar']  
tokens\_sin\_pal\_vacias = [x for x in tokens if x not in pal\_vacias]  
tokens\_sin\_pal\_vacias

[‘niño’, ‘salió’, ‘jugar’]

## Raíces y lemas

*pendiente* …

# Vectores TF-IDF

El nombre que se le da a las tablas creadas con el saco de palabras se les denomina **TF-IDF** o frecuencia de términos por la inversa de la frecuencia en el documento ( *Term Frequency times inverse document frequency* ).

### Frecuencias

Primero se obtienen las frecuencias.

### Ejemplo con artículo

Se utiliza como ejemplo el artículo de wikipedia del rehilete o molinillo

rehilete = """  
El molinillo, molinete, remolino, renglete, rehilete o reguilete es una especie de juguete compuesto por una varilla de madera a la que se clava, en la parte superior, una figura de aspas de molinillo construida con papel celofán o cartulina, habitualmente de colores llamativos. Con el viento, las aspas giran y crean efectos de color.  
  
Se le llama de muy diversas formas según los países; en España es más conocido como «molinillo»; en México o Perú, «rehilete», en Guatemala y el resto de Centroamérica y también en Cuba se le conoce como «reguilete», en Colombia se conoce como ringlete o renglete. En Chile se le conoce como «remolino».  
  
María Moliner cita los siguientes sinónimos: gallo, molinete, rehilandera, rodachina, rongigata, ventolera y voladera.  
  
Rehilete, a veces pronunciado reguilete o rejilete, proviene del verbo rehilar. Proviene de la idea del movimiento rotatorio y tembloroso del huso y de la hebra en el acto de hilar. Se usaba en el antiguo castellano o en Segovia con el sentido de temblor y posiblemente se deriva del godo "reiro" con significado de temblor o tremor. Diccionario etimológico de la lengua castellana. Dr. D. Pedro Felipe Monlau. 1881.  
"""

from collections import Counter  
from nltk.tokenize import TreebankWordTokenizer  
tokenizador = TreebankWordTokenizer()  
tokens = tokenizador.tokenize(rehilete.lower())  
token\_contador = Counter(tokens)  
token\_contador

Counter({‘el’: 6, ‘molinillo’: 3, ‘,’: 18, ‘molinete’: 2, ‘remolino’: 2, ‘renglete’: 1, ‘rehilete’: 3, ‘o’: 7, ‘reguilete’: 3, ‘es’: 2, ‘una’: 3, ‘especie’: 1, ‘de’: 14, ‘juguete’: 1, ‘compuesto’: 1, ‘por’: 1, ‘varilla’: 1, ‘madera’: 1, ‘a’: 2, ‘la’: 5, ‘que’: 1, ‘se’: 7, ‘clava’: 1, ‘en’: 10, ‘parte’: 1, ‘superior’: 1, ‘figura’: 1, ‘aspas’: 2, ‘construida’: 1, ‘con’: 4, ‘papel’: 1, ‘celofán’: 1, ‘cartulina’: 1, ‘habitualmente’: 1, ‘colores’: 1, ‘llamativos.’: 1, ‘viento’: 1, ‘las’: 1, ‘giran’: 1, ‘y’: 7, ‘crean’: 1, ‘efectos’: 1, ‘color.’: 1, ‘le’: 3, ‘llama’: 1, ‘muy’: 1, ‘diversas’: 1, ‘formas’: 1, ‘según’: 1, ‘los’: 2, ‘países’: 1, ‘;’: 2, ‘españa’: 1, ‘más’: 1, ‘conocido’: 1, ‘como’: 4, ‘«’: 4, ‘»’: 4, ‘méxico’: 1, ‘perú’: 1, ‘guatemala’: 1, ‘resto’: 1, ‘centroamérica’: 1, ‘también’: 1, ‘cuba’: 1, ‘conoce’: 3, ‘colombia’: 1, ‘ringlete’: 1, ‘renglete.’: 1, ‘chile’: 1, ‘.’: 2, ‘maría’: 1, ‘moliner’: 1, ‘cita’: 1, ‘siguientes’: 1, ‘sinónimos’: 1, ‘:’: 1, ‘gallo’: 1, ‘rehilandera’: 1, ‘rodachina’: 1, ‘rongigata’: 1, ‘ventolera’: 1, ‘voladera.’: 1, ‘veces’: 1, ‘pronunciado’: 1, ‘rejilete’: 1, ‘proviene’: 2, ‘del’: 4, ‘verbo’: 1, ‘rehilar.’: 1, ‘idea’: 1, ‘movimiento’: 1, ‘rotatorio’: 1, ‘tembloroso’: 1, ‘huso’: 1, ‘hebra’: 1, ‘acto’: 1, ‘hilar.’: 1, ‘usaba’: 1, ‘antiguo’: 1, ‘castellano’: 1, ‘segovia’: 1, ‘sentido’: 1, ‘temblor’: 2, ‘posiblemente’: 1, ‘deriva’: 1, ‘godo’: 1, ‘``’: 1, ‘reiro’: 1, "’‘": 1, ’significado’: 1, ‘tremor.’: 1, ‘diccionario’: 1, ‘etimológico’: 1, ‘lengua’: 1, ‘castellana.’: 1, ‘dr.’: 1, ‘d.’: 1, ‘pedro’: 1, ‘felipe’: 1, ‘monlau.’: 1, ‘1881’: 1})

from nltk.corpus import stopwords  
  
pal\_vacias = stopwords.words('spanish')  
  
tokens = [x for x in tokens if x not in pal\_vacias]  
rehilete\_contador = Counter(tokens)  
rehilete\_contador

Counter({‘molinillo’: 3, ‘,’: 18, ‘molinete’: 2, ‘remolino’: 2, ‘renglete’: 1, ‘rehilete’: 3, ‘reguilete’: 3, ‘especie’: 1, ‘juguete’: 1, ‘compuesto’: 1, ‘varilla’: 1, ‘madera’: 1, ‘clava’: 1, ‘parte’: 1, ‘superior’: 1, ‘figura’: 1, ‘aspas’: 2, ‘construida’: 1, ‘papel’: 1, ‘celofán’: 1, ‘cartulina’: 1, ‘habitualmente’: 1, ‘colores’: 1, ‘llamativos.’: 1, ‘viento’: 1, ‘giran’: 1, ‘crean’: 1, ‘efectos’: 1, ‘color.’: 1, ‘llama’: 1, ‘diversas’: 1, ‘formas’: 1, ‘según’: 1, ‘países’: 1, ‘;’: 2, ‘españa’: 1, ‘conocido’: 1, ‘«’: 4, ‘»’: 4, ‘méxico’: 1, ‘perú’: 1, ‘guatemala’: 1, ‘resto’: 1, ‘centroamérica’: 1, ‘cuba’: 1, ‘conoce’: 3, ‘colombia’: 1, ‘ringlete’: 1, ‘renglete.’: 1, ‘chile’: 1, ‘.’: 2, ‘maría’: 1, ‘moliner’: 1, ‘cita’: 1, ‘siguientes’: 1, ‘sinónimos’: 1, ‘:’: 1, ‘gallo’: 1, ‘rehilandera’: 1, ‘rodachina’: 1, ‘rongigata’: 1, ‘ventolera’: 1, ‘voladera.’: 1, ‘veces’: 1, ‘pronunciado’: 1, ‘rejilete’: 1, ‘proviene’: 2, ‘verbo’: 1, ‘rehilar.’: 1, ‘idea’: 1, ‘movimiento’: 1, ‘rotatorio’: 1, ‘tembloroso’: 1, ‘huso’: 1, ‘hebra’: 1, ‘acto’: 1, ‘hilar.’: 1, ‘usaba’: 1, ‘antiguo’: 1, ‘castellano’: 1, ‘segovia’: 1, ‘temblor’: 2, ‘posiblemente’: 1, ‘deriva’: 1, ‘godo’: 1, ‘``’: 1, ‘reiro’: 1, "’‘": 1, ’significado’: 1, ‘tremor.’: 1, ‘diccionario’: 1, ‘etimológico’: 1, ‘lengua’: 1, ‘castellana.’: 1, ‘dr.’: 1, ‘d.’: 1, ‘pedro’: 1, ‘felipe’: 1, ‘monlau.’: 1, ‘1881’: 1})

### Vectorizando

Luego de obtener las frecuencias, se convierte el resultado en vectores.

vector\_documento = []  
extens\_doc = len(tokens)  
  
for cve, valor in rehilete\_contador.most\_common():  
 vector\_documento.append(valor / extens\_doc)  
vector\_documento[:10]

[0.13043478260869565, 0.028985507246376812, 0.028985507246376812, 0.021739130434782608, 0.021739130434782608, 0.021739130434782608, 0.021739130434782608, 0.014492753623188406, 0.014492753623188406, 0.014492753623188406]

## Espacios vectoriales

### La ley de Zipf

### Estableciendo un modelo

### Distancia por coseno

# Análisis semántico latente

## Descomposición en valores singulares

## Análisis de componentes principales

## Análisis discriminante lineal

## Distribución de Dirichlet latente

## Comparación de resultados

# Redes neuronales

## Aprendizaje profundo

## Redes complejas

## Propagación retroactiva

## Secuencia a secuencia