Text Mining

Juan de Dios Romero Palop

AFI – Abril 2024

SOBRE MI

- Ingeniería Informática y Matemáticas en @Universidad Autónoma de Madrid
- Máster en Big Data y Data Science @AFI (1ª promoción)
- Data Scientist y Product Manager @BBVA AI Factory
- Sr. Data & AI Product Manager @Stuart


It's Your Turn!

Text Mining… ¿para qué?

Google

CENTRAL INTELLIGENCE AGENCY
UNITED STATES OF AMERICA

Microsoft

amazon

Text Mining… ¿para qué?

- Análisis de texto
- Extracción de información
- Clasificación de textos
- Búsqueda de información relevante
- *Sentiment analysis*
- Generación automática de texto
- **(new)** Manejar y entrenar Large Language Models (LLMs)
- *…*

Indice

1. BACK TO BASICS

Estructuras de datos

FICHERO/ DOCUMENTO / FILE


EL INGENIOSO
HIDALGO DON QVI-
XOTE DE LA MANCHA,
Compuesto por Miguel de Cervantes
Saauedra.
DIRIGIDO AL DVQVE DE BEIAR,
Marques de Gibraleon, Conde de Benalcaçar, y Baña-
res, Vizconde de la Puebla de Alcocer, Señor de
las villas de Capilla, Curiel, y
Burguillos.
Año, 1605.
CON PRIVILEGIO.
EN MADRID Por Iuan de la Cuesta.

Estructuras de datos

FICHERO / DOCUMENTO / FILE

↳ PÁRRAFO (separación por \n - Tecla ENTER)

“Primera parte del Ingenioso hidalgo Don Quijote de la Mancha.”

“Capítulo Primero. Que trata de la condición y ejercicio del famoso hidalgo don Quijote de la Mancha.”

“En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no hace mucho que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de más vaca que carnero, salpicón todas las noches, (…), consumían las tres partes de su hacienda.”

Estructuras de datos

FICHERO / DOCUMENTO / FILE

↳ PÁRRAFO (separación por \n)

“Primera parte del Ingenioso hidalgo Don Quijote de la Mancha.**\n**”

“Capítulo Primero. Que trata de la condición y ejercicio del famoso hidalgo don Quijote de la Mancha.**\n**”

“En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no hace mucho que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de más vaca que carnero, salpicón todas las noches, (…), consumían las tres partes de su hacienda.”

Estructuras de datos

FICHERO / DOCUMENTO / FILE

↳ PÁRRAFO (separación por \n)

↳ FRASE (separación por .)

“Primera parte del Ingenioso hidalgo Don Quijote de la Mancha.”

“Capítulo Primero.“

“Que trata de la condición y ejercicio del famoso hidalgo don Quijote de la Mancha.”

“En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no hace mucho que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. ”

Estructuras de datos

FICHERO / DOCUMENTO / FILE

- PÁRRAFO (separación por \n)
  - FRASE (separación por .)
    - PALABRA (separación por ‘ ‘)

“En”, “un”, “lugar”, “de”, “la”, “Mancha”, “de”, “cuyo”...

Estructuras de datos

FICHERO / DOCUMENTO / FILE

↳ PARRAFO (separación por \n)

↳ FRASE (separación por .)

↳ PALABRA (separación por ‘ ‘)

↳ CARACTER (≠ LETRA)

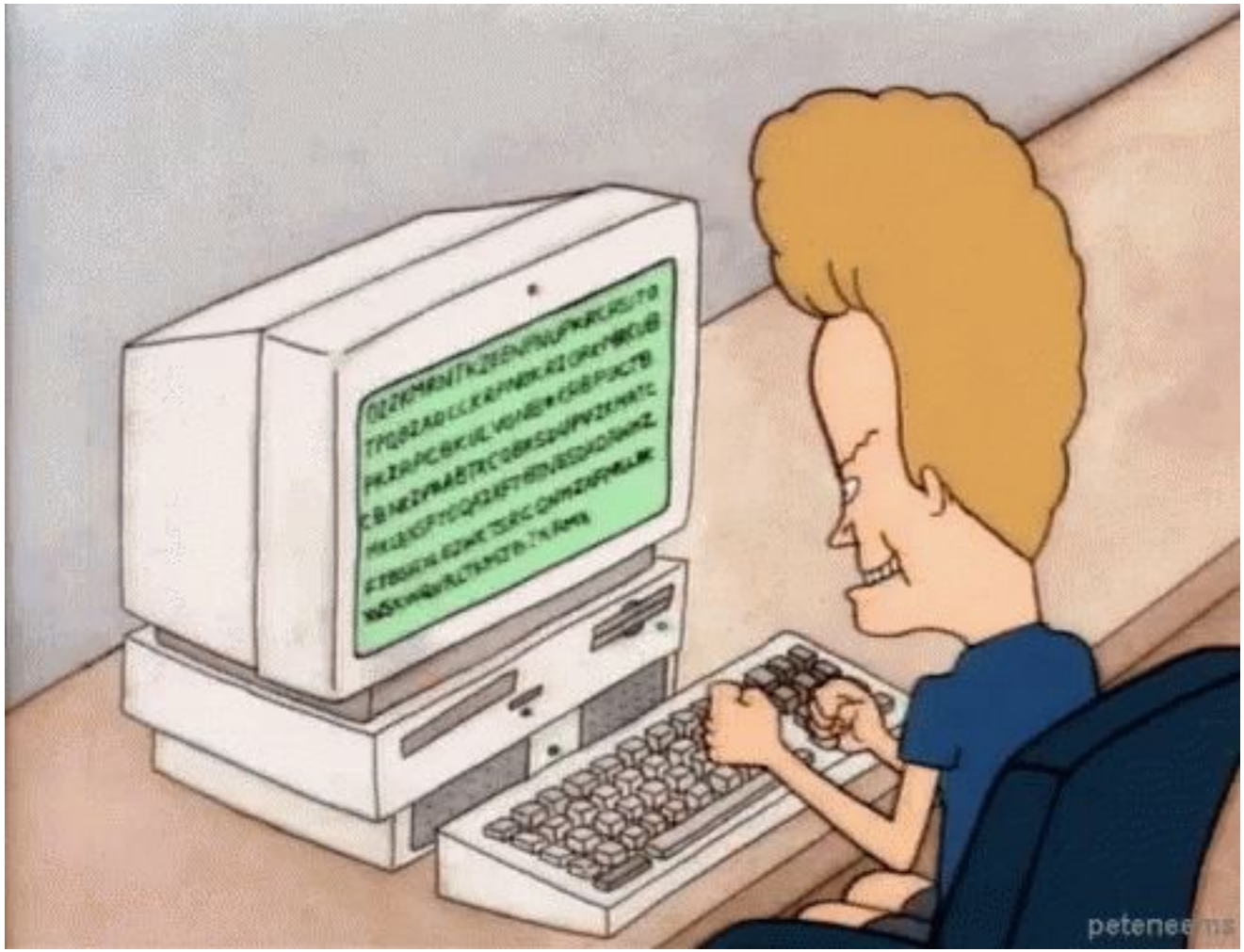
“E”, “n”, “ “, “u“, “n”, “ ”, “l”, “u”, “g”, “a”, “r”…

Caracteres ASCII

Caracteres ASCII de control		
00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sinc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles					
32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp



Expresiones regulares

Las expresiones regulares (regex o regexp) son una herramienta común a todos los lenguajes de programación y sirven para realizar búsquedas no específicas pero regulares dentro de textos.

Son una de las herramientas más potentes y más utilizadas a la hora de extraer información, realizar comprobaciones o desarrollar productos de datos.

Expresiones regulares

Complementan las funciones vistas en el apartado anterior multiplicando su capacidad.

Ejemplos de uso:

- Buscar todas las palabras que empiezan por mayúscula dentro de un texto sin tener que especificar todas las letras.
- Buscar fechas.
- Buscar distintas apariciones del mismo verbo.

Expresiones regulares - básicos

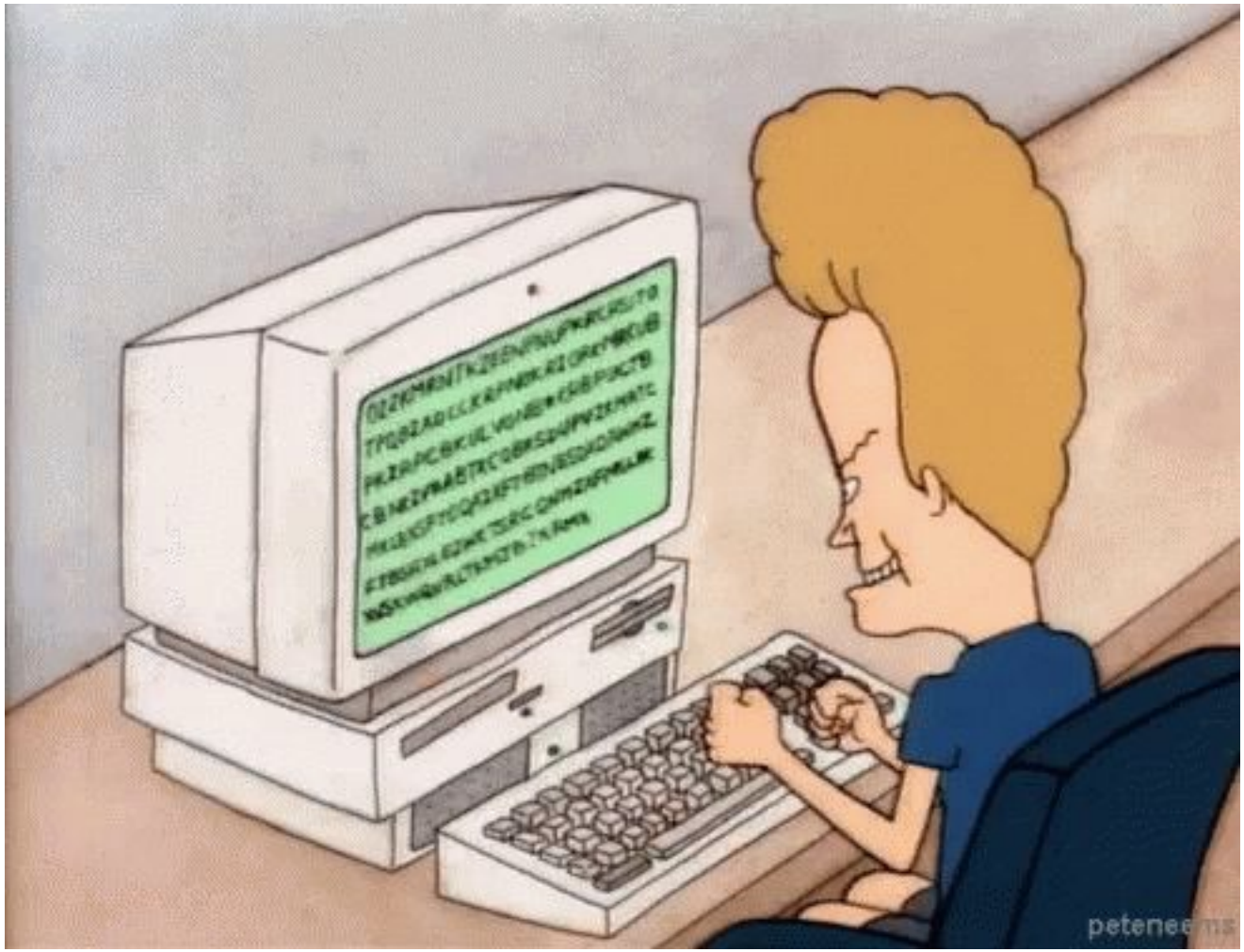
.	Un carácter cualquiera.
^	Inicio de string.
$	Final de string.
[…]	Caracteres incluidos dentro.
a-z	Letras minúsculas.
A-Z	Letras mayúsculas.
0-9	Dígitos.
[^abc]	Cualquier carácter menos los que van después.
a|b	O un carácter u otro.

Expresiones regulares - grupos

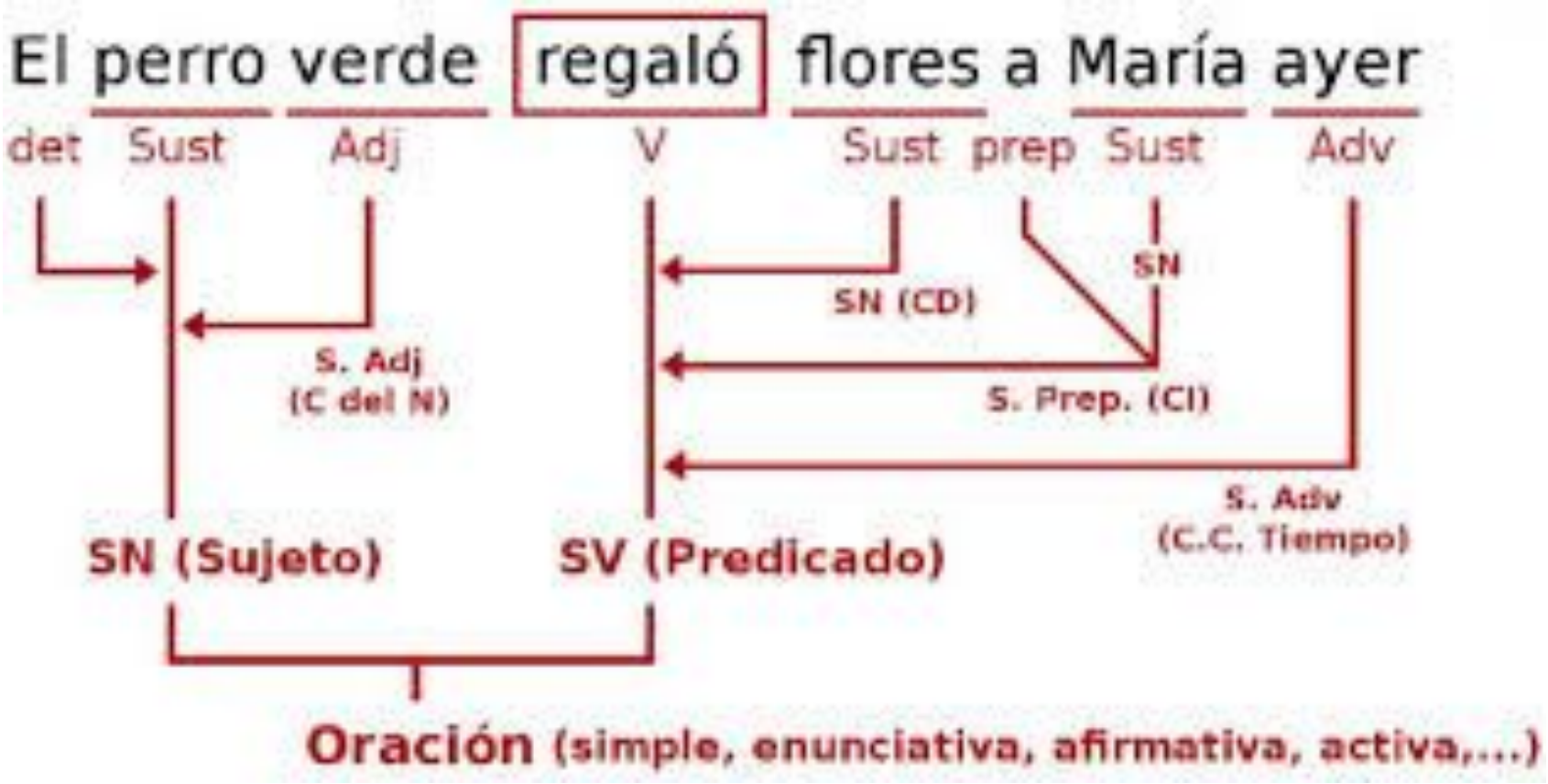
\	Escape de caracteres.
\b	Limite de palabra.
\d	Dígito.
\D	Cualquier carácter menos un dígito.
\s	Cualquier blanco (‘ ‘, tabulador, salto de linea)
\S	Cualquier carácter menos los blancos.
\w	Cualquier valor alfanumérico.
\W	Cualquier carácter menos un alfanumérico.

Expresiones regulares - cuantificadores

*	0 o más veces.
+	1 o más veces.
?	1 vez o ninguna.
{n}	Exactamente *n* veces.
{n,}	Al menos n veces.
{,n}	Como mucho *n* veces.
{n,m}	Al menos *n* veces y como mucho *m* veces.



2. LENGUA, 3º ESO



NLTK

- *Natural Language Tool Kit*
- Modelos pre-entrenados en inglés, capacidad de re-entrenar con corpus anotados de otros idiomas.

✓ Funciones **morfológicas** *e.g. Stemming/Lemmatization*

✓ Funciones **semánticas** *e.g. Sinónimos/Antónimos*

✓ Funciones **sintácticas** *e.g. Part Of Speech (POS)*

SpaCy

- Gran escalabilidad y rendimiento.
- Soporte directo a múltiples idiomas.

✓ Funciones **morfológicas** *e.g. Stemming/Lemmatization*

✓ Funciones **semánticas** *e.g. Sinónimos/Antónimos*

✓ Funciones **sintácticas** *e.g. Part Of Speech (POS)*

Tokenización

- Primer paso para poder aplicar otras funciones.
- Dividir texto en estructuras más simples.
- *Sentence tokenizer*: división en frases. Función por defecto y posibilidad de adaptación a las estructuras de distintos idiomas.
- *Word tokenizer*: división en palabras.
- Se mantienen los signos de puntuación en los tokens.
- [*SpaCy*] Mantiene los saltos de línea también.

Part Of Speech

- Ambas librerías proporcionan una función que, dada una frase, asigna a cada una de las palabras una etiqueta de la función sintáctica que realiza en la frase. Además incluyen visualizaciones de los resultados.

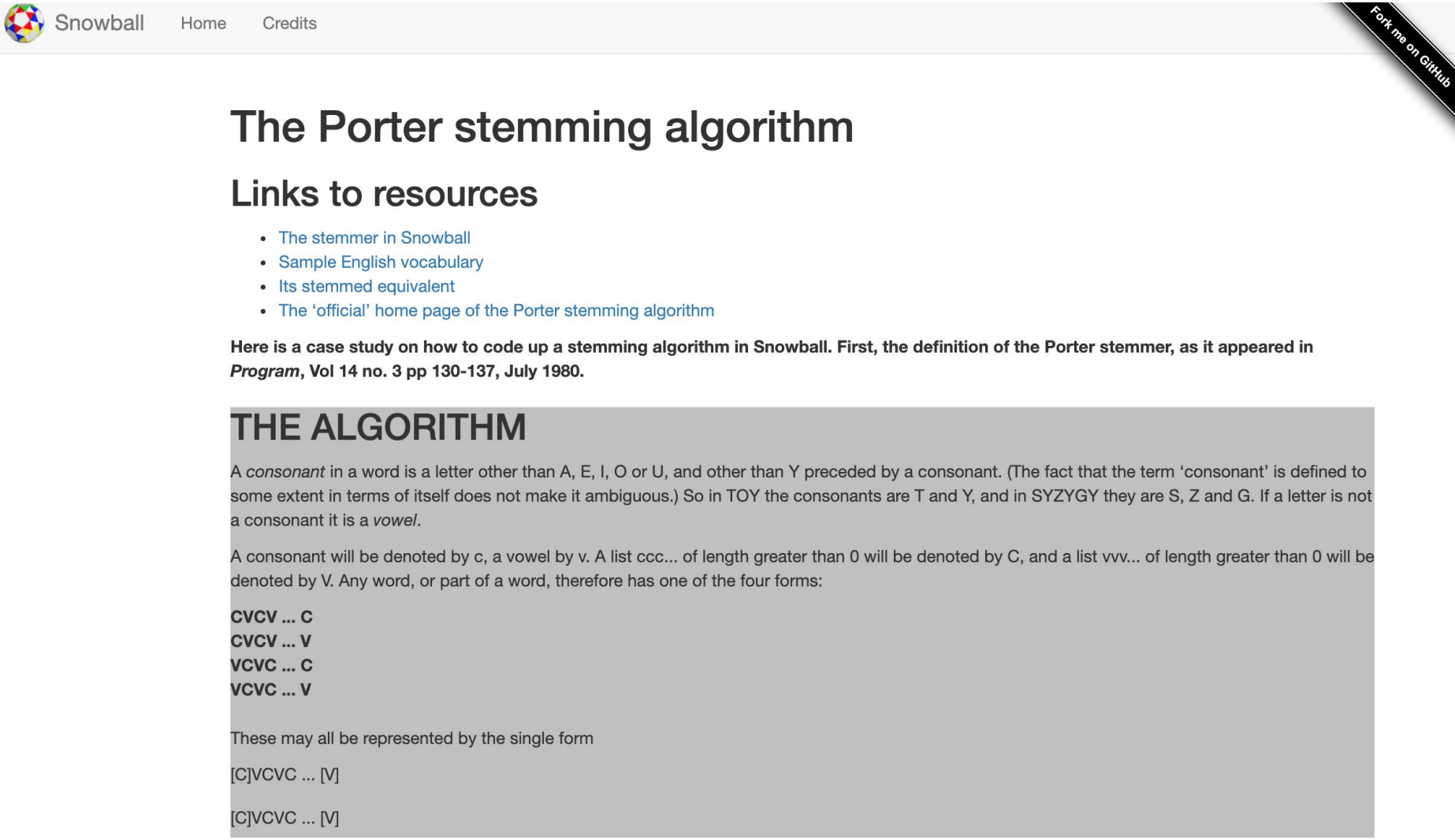
- [NLTK] Por defecto, tiene etiquetador entrenado en inglés. Existe la opción de entrenar el etiquetador con distintos corpus anotados para adaptar la herramienta a distintas sintaxis. Distintos idiomas, lenguajes de programación…

- [SpaCy] Etiquetador ya entrenado en castellano.

Abbreviation	Meaning
CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list market
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)

Stemming (sólo en NLTK)

- El proceso de stemming consiste en reducir las palabras a su raíz. El resultado es útil para realizar conteos, entrenar modelos, etc.
- El resultado no siempre (de hecho, pocas veces) es una palabra válida. Importante de cara a pérdida de información.
- Para llevar a cabo un proceso de stemming no se tiene en cuenta el contexto. No se tiene en cuenta la función que juega la palabra en una frase.
- Existen varios algoritmos de stemming. El más popular es el de Porter.



https://snowballstem.org/algorithms/porter/stemmer.html

Lemmatizacion

- El objetivo es el mismo que el de aplicar *stemming:* encontrar la raíz de la palabra.
- Sin embargo, en lemmatizacion el resultado siempre deber ser una palabra existente.
- Para llevar a cabo un proceso de lemmatizacion es útil tener en cuenta su contexto en la frase. Una misma palabra puede devolver distintos resultados dependiendo de esto.
- [SpaCy] Comete algunos errores por no tener en cuenta el contexto. [NLTK] Puede incluirse y lo utiliza.

Named Entities Recognition (NER)

- *NER* es el proceso de encontrar dentro del texto palabras o grupos de palabras que identifican a personas, lugares, empresas, etc.
- Más adelante se ha extendido a números, medidas, fechas y otro tipo de expresiones incluyendo cifras.
- Este tipo de referencias suelen ser muy informativas a la hora de analizar el contenido de un texto. Son de gran ayuda para clasificar textos.

Stopwords

- El término *stopwords* hace referencia a aquellas palabras que sirven de unión dentro de las frases pero que no tienen significado propio.
- *E.g.* El, a, aquellas, que, de, las, que.
- Es importante manejarlas correctamente a la hora de trabajar con los textos ya que pueden tener una gran influencia por su elevada frecuencia.
- Existen conjuntos predefinidos de *stopwords* en distintos idiomas pero a veces es necesario adaptarlos al problema particular.

Text @NLTK

- NLTK define una estructura de datos llamada *Text* que implementa varias funciones para realizar búsquedas dentro del texto.
- Son funciones muy útiles para manejar textos grandes. Pueden servir incluso para analizar el estilo de la persona que ha escrito el texto.
- NLTK incluye varios ejemplos de textos dentro de la librería books.
- Funciones: concordance, similar, common_contexts, dispersion_plot, collocation.

Wordnet @NLTK

- Word Net es una librería de funcionalidades incluida dentro de NLTK.
- A partir de un corpus, Word Net proporciona funcionalidades relacionadas con la semántica de las palabras.
- Definiciones, sinónimos, antónimos, frases de ejemplo…
- Da soporte a múltiples idiomas pero siempre con el inglés como idioma central.

Extra info @spaCy

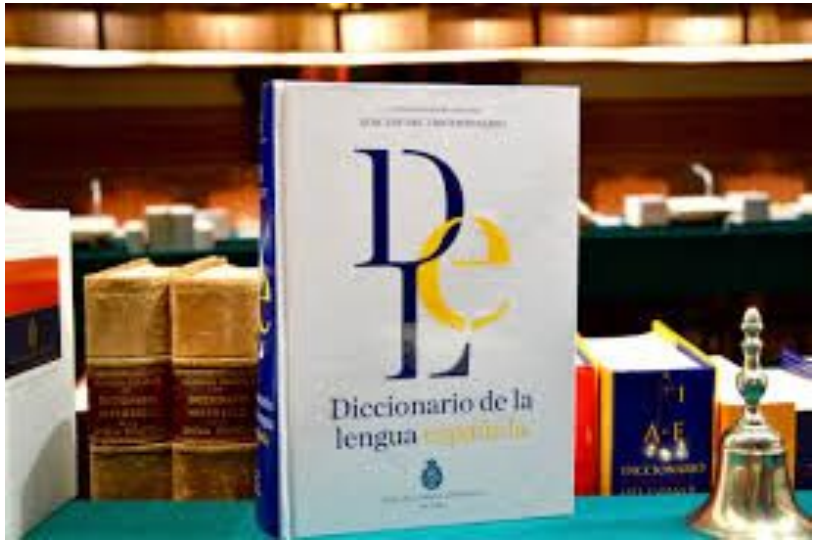
- En el momento en que se lee un texto con SpaCy automáticamente se procesa y se llevan a cabo todos los análisis.
- Cada análisis resulta en un etiquetado de los tokens que puede consultarse.
- Un ejemplo sería el análisis de los conjuntos de nombres (noun_chunks) que analiza las relaciones de los sustantivos con el resto de elementos de la frase.
- Mantiene el soporte a múltiples idiomas.

3. DISTANCIAS


LITTLE

¿Cómo desarrollarías un corrector de escritura en móvil en media hora?

¿Cómo desarrollarías un corrector de escritura en móvil en media hora?



\+ Función de distancia entre palabras

Distancia de Hamming

Nº de posiciones en las que tenemos el mismo símbolo en ambas cadenas.

Esta distancia solo sirve para cadenas de la misma longitud.

Cuanto más alta más cercanas.

Hamming(lona, lobo) = 2

Hamming(loba, bola) = 2

Hamming(losa, lonas) = NO APLICA

Distancia de Levenshtein

Nº mínimo de inserciones, borrados o modificaciones de caracteres para convertir una cadena en otra.

Cuanto más baja, más cercanas son las palabras.

Lev(lona, lobo) = 2

Lev(loba, bola) = 2

Lev(losa, lonas) = 2

Distancia de Damerau-Levenshtein

Igual que la de Levenshtein pero una trasposición de símbolos adyacentes sólo incrementa en uno la distancia.

DLev(lona, lobo) = 2

DLev(loba, bola) = 2

DLev(loba, loab) = 1

DLev(losa, lonas) = 2

Distancia de q-gramas

Número total de q-gramas que aparecen en ambas cadenas.

Cuantos más, más cercanas.

2-gramas(lona, lobo) = 2 (lo, on, na, lo, ob, bo)

2-gramas(loba, bola) = 0 (lo, ob, ba, bo, ol, la)

2-gramas(losa, lonas) = 2 (lo, os, sa, lo, on, na, as)

Otras distancias

- Distancia del coseno
- Distancia de Jaccard
- Distancia de Jaro-Winkler
- …

4. TODO FUNCIONA MEJOR CON NÚMEROS

¿dist(malo, bueno) > dist(malo, mago)?

¿dist(malo, bueno) > dist(malo, mago)?

Las distancias del apartado anterior sirven para corregir *typos*, intentar predecir qué palabra está escribiendo el usuario una vez ha empezado a escribir o trabajar con entidades propias.

Pero, ¿sirven para otro tipo de problemas como la clasificación o la generación de textos?

¿En qué casos creéis que la respuesta a la pregunta de las distancias es sí y en cuales es no?

De textos a vectores

Todos los modelos de Machine Learning están basados en el manejo de vectores y matrices de números.

¿Qué hacemos cuando queremos trabajar con textos? Convertir los textos a vectores o matrices que los representen: el objetivo es que dos textos que tienen cosas en común deben generar vectores que estén cercanos.

Vectorización aditiva

La mayoría de algoritmos de vectorización de textos son aditivos. Es decir, el vector que representa a un texto se genera a partir de la suma de los vectores que representan a sus elementos: palabras, n-gramas, parejas de palabras…

me = (rojo, azul, blanco);

gusta= (blanco, blanco, blanco);

mucho=(blanco, amarillo, blanco);

“me gusta mucho” = (rojo, verde, blanco)

Preprocesado

- Limpieza de texto
- Tokenización
- Normalización
- *Lemmatización/Stemming*

Bag of words

- Es la manera más simple de vectorizar textos.
- Se generan vectores del tamaño del vocabulario del dataset.
- Cada palabra del vocabulario tiene asignada una posición y el vector correspondiente a esa palabra tiene un 1 en esa posición y 0s en el resto.
- A cada texto del dataset se le asigna el vector aditivo de los elementos que lo forman.

Bag of words

Dataset

T1: ”Me ha gustado la película”

T2: “No me ha gustado”

T3: “La película es mala”

Bag of words

Dataset

T1: ”Me ha gustado la película”

T2: “No me ha gustado”

T3: “La película es mala”

Vocabulario

me = 1; haber = 2; gustar= 3; la=4; película=5; no=6; ser=7; mala=8

Bag of words

Dataset

T1: ”Me ha gustado la película”

T2: “No me ha gustado”

T3: “La película es mala”

Vocabulario

me = 1; haber = 2; gustar= 3; la=4; película=5; no=6; ser=7; mala=8

Vectorización

T1 = [1,1,1,1,1,0,0,0]

T2 = [1,1,1,0,0,1,0,0]

T3 = [0,0,0,1,1,0,1,1]

TF-IDF

- *Term Frequency-Inverse Document Frequency*.
- Se generan vectores del tamaño del vocabulario del dataset.
- Al contrario que bag of words, TF-IDF le da distintos valores a cada palabra según su aporte de información para hacer un texto único. El valor depende de la frecuencia con la que aparece la palabra en el dataset.
- TF-IDF = TF * IDF.

TF

- *TF* = Nº de veces que la palabra aparece en el texto.
- Aunque esta es la definición básica existen otras opciones para calcular el término TF.
  - Booleano: 1 si la palabra aparece, 0 si no.
  - Frecuencia normalizada: se divide por el tamaño del texto o por la frecuencia de la palabra que más aparece en el texto.
  - Escalado logarítmico: log(1 + TF)

IDF

- *IDF :=* cuánto ayuda esa palabra a diferenciar textos = log(Nº de textos totales / Nº de textos que incluyen esa palabra)
- A veces se modifica la fórmula básica para protegerse ante la aparición de nuevas palabras que podrían generar divisiones por 0.
  - Ajuste del denominador: log(Nº de textos/1+Nº textos incluyen)
- Palabras raras => IDF alto,
- Palabras comunes => IDF ≈ 0

TF-IDF: ejemplo

TEXTO 1

Término	Frecuencia
gustar	7
película	8
buena	3

TEXTO 2

Término	Frecuencia
gustar	8
película	11
mala	3
no	8

TEXTO 3

Término	Frecuencia
película	4
Tarantino	1
Oscar	1

TF-IDF: ejemplo

TEXTO 1

Término	Frecuencia	Frec. normalizada
gustar	7	7/8
película	8	8/8
buena	3	3/8

TEXTO 2

Término	Frecuencia	Frec. normalizada
gustar	8	8/11
película	11	11/11
mala	3	3/11
no	8	8/11

TEXTO 3

Término	Frecuencia	Frec. normalizada
película	4	4/4
Tarantino	1	1/4
Oscar	1	1/4

TF-IDF: ejemplo

TEXTO 1

Término	Frecuencia	Frec. normalizada	IDF
gustar	7	7/8	log(3/2)
película	8	8/8	log(1) = 0
buena	3	3/8	log(3)

TEXTO 2

Término	Frecuencia	Frec. normalizada	IDF
gustar	8	8/11	log(3/2)
película	11	11/11	log(1)=0
mala	3	3/11	log(3)
no	8	8/11	log(3)

TEXTO 3

Término	Frecuencia	Frec. normalizada	IDF
película	4	4/4	log(1)=0
Tarantino	1	1/4	log(3)
Oscar	1	1/4	log(3)

TF-IDF: ejemplo

TEXTO 1

Término	Frecuencia	Frec. normalizada	IDF	TF-IDF
gustar	7	7/8	log(3/2)	0.15
película	8	8/8	log(1) = 0	0
buena	3	3/8	log(3)	0.17

TEXTO 2

Término	Frecuencia	Frec. normalizada	IDF	TF-IDF
gustar	8	8/11	log(3/2)	0.13
película	11	11/11	log(1)=0	0
mala	3	3/11	log(3)	0.13
no	8	8/11	log(3)	0.34

TEXTO 3

Término	Frecuencia	Frec. normalizada	IDF	TF-IDF
película	4	4/4	log(1)=0	0
Tarantino	1	1/4	log(3)	0.12
Oscar	1	1/4	log(3)	0.12

TF-IDF

Dataset

T1, T2, T3

Vocabulario

gustar = 1; película = 2; buena = 3; mala = 4; no = 5; Tarantino = 6; Óscar = 7

Vectorización

T1 = [0.15,0,0.17,0,0,0,0]

T2 = [0.13,0,0,0.13,0.34,0,0]

T3 = [0,0,0,0,0,0.12, 0.12]

TF-IDF: ¡ojo con el split de los datos!

Uno de los errores más comunes que se cometen al aplicar TF-IDF es mezclar los conjuntos de entrenamiento y test.

Al igual que ocurre con el resto de datos, la información necesaria para aplicar TF-IDF debe provenir únicamente de los datos de entrenamiento.

El proceso a seguir es el siguiente: se calcula el término IDF para todas las palabras del conjunto de entrenamiento y se guardan para utilizarlas tanto en entrenamiento como en la fase de inferencia. Nunca se recalculan con el conjunto de test.

Word embbedings

- Tanto *bag of words* como *TF-IDF* tienen en cuenta las palabras incluidas en el vocabulario pero no tienen en cuenta el contexto en el que se utilizan.

- Existen una serie de algoritmos de vectorización conocidos como *word embeddings* que utilizando la potencia de las redes neuronales generan representaciones vectoriales de las palabras analizando el contexto en el que se utilizan.

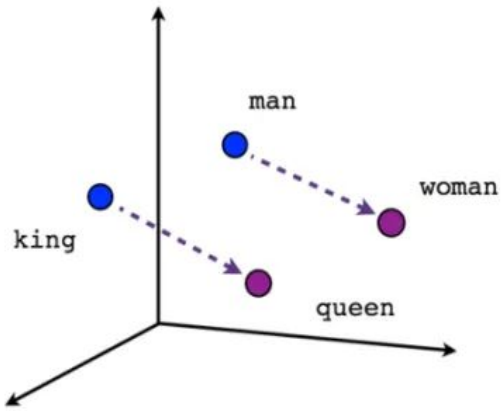
Word embbedings

- Los *word embeddings* aseguran que las representaciones vectoriales de dos palabras que se utilicen en contextos parecidos estarán cercanas en el espacio vectorial definido.
- Para poder entrenar estos algoritmos hacen falta datasets muy grandes que permitan generalizar y sacar conclusiones sobre los contextos de cada palabra.
- Pueden ser entrenados con datasets de distintos ámbitos.

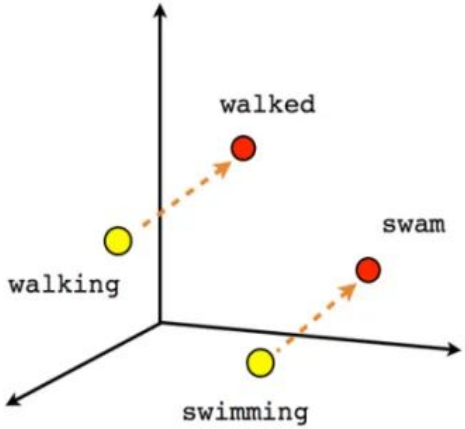
word2vec

- Creado en 2013 por un grupo de investigadores de Google.
- Se basa en una red neuronal de 2 capas y está entrenado con datos de Google.
- Existen dos modos de representar el problema que se traducen en dos arquitecturas de red distintas: se conocen con *Skip-Gram* o *Continuous Bag of Words* (CBOW) .
- En ambos casos el resultado es la asignación de una representación vectorial a cada palabra del vocabulario.

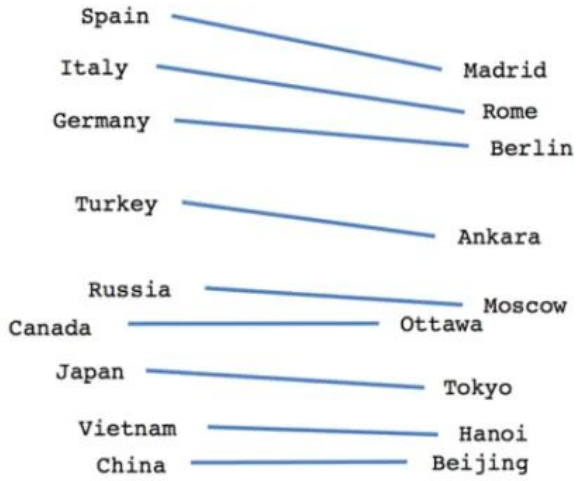
word2vec



Male-Female



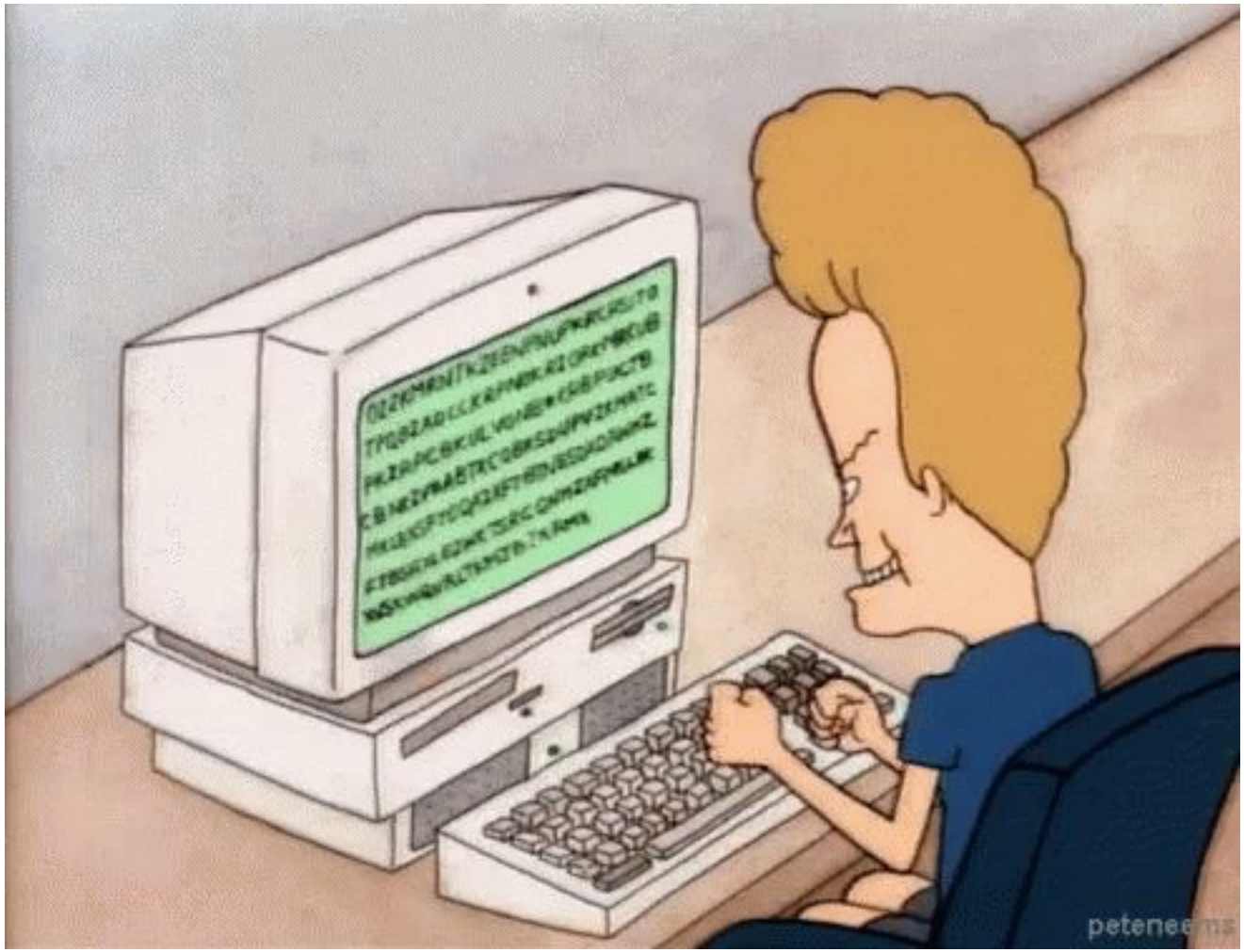
Verb tense



Country-Capital

Otros modelos de word embeddings

- Stanford GloVe: https://nlp.stanford.edu/projects/glove
- Facebook fastText: https://fasttext.cc
- Google BERT: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#what-is-bert



5. CASO DE USO: ANÁLISIS DE SENTIMIENTO


ANY QUESTIONS?


That's all Folks!