Natural Language Processing

Jonnatan Arias Garcia – jonnatan.arias@utp.edu.co – jariasg@uniquindio.edu.co

David Cardenas peña - dcardenasp@utp.edu.co

Hernán Felipe Garcia - hernanf.garcia@udea.edu.co

Contenido

- **Introducción**
- **Aplicaciones**
- **Desafíos y limitaciones**

1. **Procesamiento de Texto Básico**
    1. Tokenización y segmentación de texto.
    2. Lematización y stemming.
    3. Stop Words
2. **Modelos**
    1. Representación de texto
        1. BOW
        2. TF-IDF
        3. Word Embedding
    2. Redes neuronales
    3. Red neuronal convolucional

Redes Recurrentes y lstm

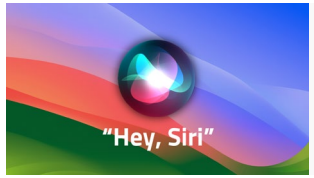
Modelos de lenguaje avanzados

GPT, Gilbert Bert

Procesamiento de Lenguaje Natural

- Es un campo de la Inteligencia Artificial
- Investiga la manera de comunicar las máquinas con las personas mediante el uso de lenguas naturales, como el español, el inglés o el chino.

- Virtualmente, cualquier lengua humana puede ser tratada por los ordenadores.
- Lógicamente, limitaciones de interés económico o práctico hace que solo las lenguas más habladas o utilizadas en el mundo digital tengan aplicaciones en uso.



Siri (20) Habla

Google Assistant (8).

Google Translate supera el centenar…

Hay entre 5000 y 7000 lenguas en el mundo.

Procesamiento de Lenguaje Natural

- Las lenguas humanas pueden expresarse por escrito (**texto**), oralmente (**voz**) y también mediante signos.

Naturalmente, el PLN está más avanzado en el tratamiento de textos, donde hay muchos más datos y son más fáciles de conseguir en formato electrónico.

Los audios, aunque estén en formato digital, hay que procesarlos para transcribirlos en letras o caracteres y, a partir de ahí, entender la pregunta. El proceso de respuesta es el inverso: primero se elabora la oración y luego se "sintetiza la voz".

Aplicaciones del NLP

1. Búsqueda de Información
2. Asistentes Virtuales y Chatbots
3. Análisis de Sentimientos
4. Traducción Automática
5. Reconocimiento de Voz
6. Resumen Automático de Texto
7. Clasificación de Texto y Categorización de Contenido
8. Generación de Texto

Desafíos y limitaciones del NLP

Desafíos

1. Ambigüedad Lingüística
2. Variabilidad del lenguaje
3. Entendimiento del Contexto
4. Coherencia en la generación de texto
5. Sesgo y privacidad de datos

Desafíos y limitaciones del NLP

Limitaciones

1. Dependencia de los datos
2. Sesgo
3. Comprensión limitada de sentido común
4. Interpretabilidad y transparencia
5. Escasez de Recursos para idiomas menos comunes

Procesamiento de textos I

¿Qué es el texto?

Secuencia de caracteres, palabras, frases y entidades nombradas, de oraciones, párrafos. Etc.

Texto -> Secuencia de palabras

Palabras -> Secuencia de letras

Palabras + símbolos de puntuación, interrogación y exclamación + Significado y sentido = Texto

Procesamiento de textos II

Consideremos las siguientes secuencias:

- lalaajradeaatengguesosedyartásobremes
- tengosedylajarradeaguaestásobrelamesa
- no tengo la jarra y la sed de agua está sobre mesa
- no tengo sed y la jarra de agua está sobre la mesa
- no, tengo sed y la jarra de agua está sobre la mesa

Mismas letras? Mismas Palabras? Orden? Puntuación?

Todo va ligado a un sentido y pude variar con cualquier orden, palabra o puntuación:

Tambien hay palabras compuestas y variaciones propios del idioma.

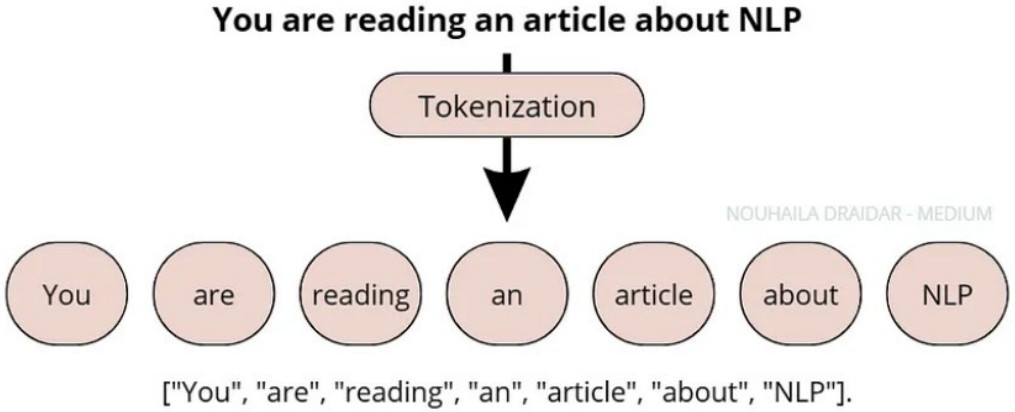
rechtsschutzversicherungsgesellschaften que significa “compañía de seguros que provee protección legal”

¿Cómo entrenar a la computadora para manejar todo esto?

Procesamiento de textos III - Token

Tokenización: Convertir caracteres, palabras o parrados en inputs para nuestro pc.

Token es la unidad de procesamiento semántico



NATURAL LANGUAGE PROCESSING USING NLTK

PYTHON

```
from nltk.tokenize import WordPunctTokenizer
from nltk.tokenize import TreebankWordTokenizer
texto = "¿Cuánto tiempo pasó desde que comí una manzana?"
texto_tokenizado1 = WordPunctTokenizer().tokenize(texto)
texto_tokenizado2 = TreebankWordTokenizer().tokenize(texto)
print(texto_tokenizado1)
# Output: ['¿', 'Cuanto', 'tiempo', 'pasó', 'desde', 'que', 'comí', 'una', 'manzana', '?']
print(texto_tokenizado2)
# Output: ['¿Cuanto', 'tiempo', 'pasó', 'desde', 'que', 'comí', 'una', 'manzana', '?']
```

Procesamiento de textos IV - Token

1. **word_tokenize**: Tokenizador de palabras estándar que divide el texto en palabras individuales.
2. **sent_tokenize**: Tokenizador de frases que divide el texto en oraciones individuales.
3. **RegexpTokenizer**: Tokenizador que utiliza expresiones regulares personalizadas para dividir el texto en tokens.
4. **WhitespaceTokenizer**: Tokenizador que divide el texto en función de los espacios en blanco.
5. **TreebankWordTokenizer**: Tokenizador que sigue las convenciones de tokenización utilizadas en el Penn Treebank.
6. **WordPunctTokenizer**: Tokenizador que divide el texto en palabras y puntuación.

Procesamiento de textos V - Token

7. PunktSentenceTokenizer: Tokenizador de frases basado en reglas que está entrenado para trabajar en varios idiomas.

8. BlanklineTokenizer: Tokenizador que divide el texto en párrafos basándose en líneas en blanco.

9. LineTokenizer: Tokenizador que divide el texto en líneas.

10. SpaceTokenizer: Tokenizador que divide el texto en función de los espacios en blanco.

11. TweetTokenizer: Tokenizador especializado para manejar tweets que tiene en cuenta las convenciones de abreviaturas y emojis comunes en Twitter.

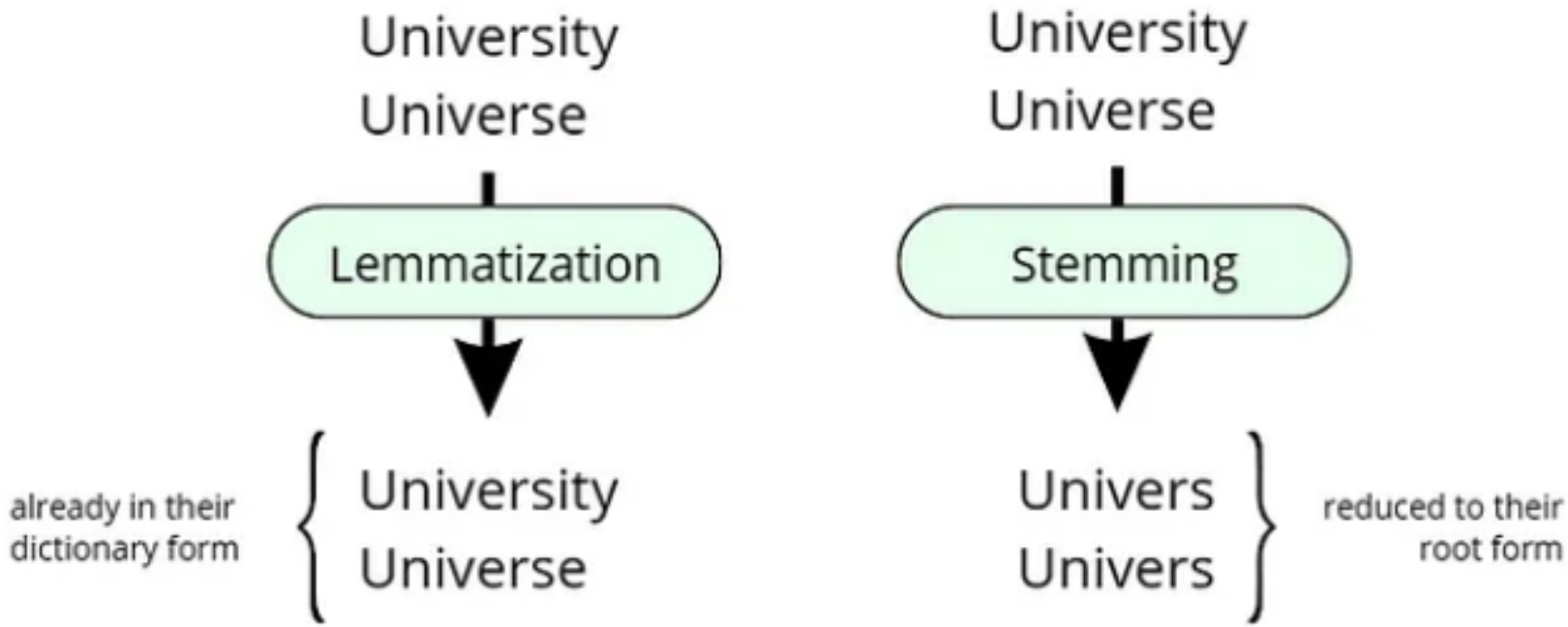
12. MWETokenizer: Tokenizador de expresiones multi-palabra (multi-word expressions) que permite tratar conjuntos de palabras como tokens únicos.

Procesamiento de textos VI

Es Habitual encontrar palabras que tengan el mismo significado pero representado de varias maneras: Plural, singular, tiempos verbales…

```
from nltk.tokenize import WordPunctTokenizer
texto_pablito = "Pablito clavó un clavito cuantos clavitos
clava pablito"
pablito_tokenizado =
WordPunctTokenizer().tokenize(texto_pablito)
print(pablito_tokenizado)
# Output: ['Pablito', 'clavó', 'un', 'clavito', 'cuantos',
'clavitos', 'clava', 'pablito']
```

Por ejemplo: clavó y clava, clavito y clavitos y Pablito y pablito.


University
Universe
Lemmatization
already in their dictionary form
University
Universe
University
Universe
Stemming
Univers
Univers
reduced to their root form

Procesamiento de textos VII - Steam

Stemming: Consiste en quitar y reemplazar los sufijos de la raíz de la palabra.

Suele ser un poco torpe y brusco, además se debe tener en cuenta que cada idioma tiene sus reglas gramaticales.

Una visión general es reemplazar o quitar terminaciones como ar, er, ir, ía, en, es, etc.

Ejemplo:

Caminar -> Camina

Easily -> Easy, easili

http://snowball.tartarus.org/algorithms/spanish/stemmer.html

Procesamiento de textos VIII - Steam

```
from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer('spanish')
stemmed_text = [stemmer.stem(i) for i in pablito_tokenizado]

print(stemmed_text)
# Output: ['pablit', 'clav', 'un', 'clavit', 'cuant', 'clavit', 'clav', 'pablit']
```

1. **PorterStemmer**: Elimina sufijos para buscar la raíz. (rápidamente-> rapidament)
2. **LancasterStemmer:** Mas agresivo que el Porter (rápidamente-> rapid)
3. **SnowballStemmer:** (porter2) mas eficiente y preciso que porter, además mejor comprensión multi-idioma. (volar -> vol)
4. **RegexpStemmer:** Utiliza expresiones regulares personalizadas.(volar -> volar)

Procesamiento de textos IX - Lemma

Lemmatización: Implica hacer un análisis del vocabulario y su morfología para retornar la forma básica de la palabra (sin conjugar, en singular,…)

Mas compleja que stem y busca generar una mejor identificación de cada palabra.

```
import stanza
stanza.download("es")
nlp = stanza.Pipeline(lang='es',
processors='tokenize,mwt,pos,lemma')
texto_pablito = "Pablito clavó un clavito cuantos clavitos
clava pablito"
doc = nlp(texto_pablito)
print(*[f'Palabra: {word.text+" "}\tLemma: {word.lemma}' for
sent in doc.sentences for word in sent.words], sep='\n')
```

```
Output:
Palabra: Pablito Lemma: Pablito
Palabra: clavó Lemma: clavar
Palabra: un Lemma: uno
Palabra: clavito Lemma: clavito
Palabra: cuantos Lemma: cuanto
Palabra: clavitos Lemma: clavito
Palabra: clava Lemma: clavar
Palabra: pablito Lemma: pablito
```

Procesamiento de textos X

StopWords

"stop words" (palabras vacías o palabras de parada) son palabras comunes que generalmente se filtran o se eliminan del texto durante el preprocesamiento.

Estas palabras son muy frecuentes en un idioma dado pero no aportan mucho significado semántico al texto.

Algunos ejemplos comunes de stop words en inglés incluyen "the", "is", "and", "of", etc.

Otros variantes al procesamiento de textos va ligado al reemplazo de sinónimos y antónimos

Modelos para PLN

Tratar computacionalmente una lengua implica un proceso de **modelamiento matemático**.

- Los ordenadores solo entienden de **bytes y dígitos** y los informáticos codifican los programas empleando lenguajes de programación.

- Los lingüistas computacionales se encargan de la tarea de "preparar" el modelo lingüístico para que los ingenieros informáticos lo implementen en un código eficiente y funcional.

Modelos para PLN

Básicamente, existen dos aproximaciones generales al problema de la modelización lingüística:

- Modelos Lógicos: modelan gramáticas

- Modelos probabilísticos de lenguaje natural: Basados en datos.

Modelos para PLN

Modelos Lógicos: modelan gramáticas

- **Análisis Sintáctico y Semántico**: Buscan comprender la estructura y el significado de las oraciones.
- **Útiles** en procesamiento de preguntas y respuestas, inferencia de relaciones, resolución de problemas lógicos.
- **Generación de Lenguaje Natural**: se puede usar para Garantizar que oraciones cumplan restricciones lógicas o semánticas

Modelos probabilísticos de lenguaje natural: Basados en datos.

- **Modelado**: Estiman la probabilidad de secuencia de palabras
- **Útiles** en clasificación de textos (análisis de sentimientos, detectar spam, traductor automático…)
- **Generación de Lenguaje Natural**: genera buenas secuencias de palabras maximizando la entrada (chatbots)

Componentes del PLN

Puntos de análisis

1. **Análisis morfológico o léxico**. Análisis interno de las palabras que forman oraciones para extraer lemas, rasgos flexivos, unidades léxica compuestas. Es esencial para la información básica: categoría sintáctica y significado léxico.

2. **Análisis sintáctico**. Consiste en el análisis de la estructura de las oraciones de acuerdo con el modelo gramatical empleado (lógico o estadístico).

Componentes del PLN

3. **Análisis semántico**. Proporciona la interpretación de las oraciones, una vez eliminadas las ambigüedades morfosintácticas.

4. **Análisis pragmático**. Incorpora el análisis del contexto de uso a la interpretación final. Aquí se incluye el tratamiento del lenguaje figurado (metáfora e ironía) como el conocimiento del mundo específico necesario para entender un texto especializado.

Los puntos de análisis se usan dependiendo del contexto.

Por ejemplo, un conversor de texto a voz no necesita el análisis semántico o pragmático. Pero un sistema conversacional requiere información muy detallada del contexto y del dominio temático.

Representación de texto

Bag Of Words

Term Frequency-Inverse Document Frequency (TF-IDF)

Word Embeddings

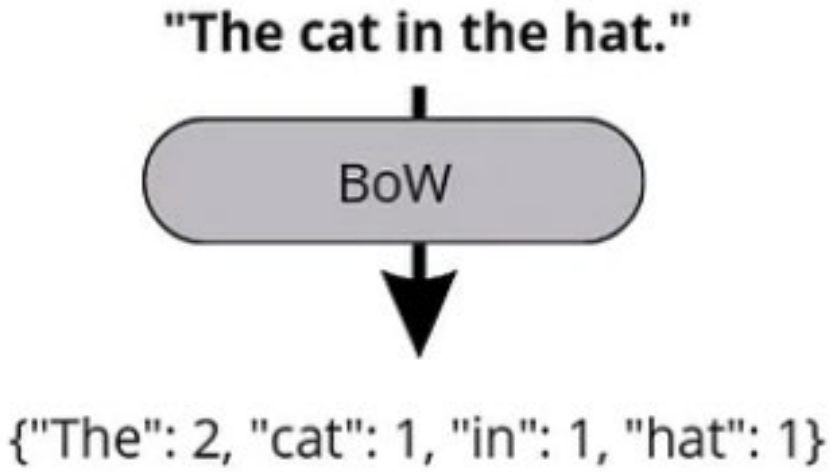
Bolsa de Palabras BoW

La manera para que el PC pueda interpretar palabras se llama **Vectorización de texto** y consiste en convertir oraciones de la siguiente forma:

	esta	pelicula	es	malisima	buenisima
Esta pelicula es buenisima	1	1	1	0	1
Esta pelicula es malisima	1	1	1	1	0
malisima	0	0	0	1	0

Se construyo una matriz con cada palabra tokenizada, donde cada fila se completa con 0 o 1 según la aparición o ausencia. A estos vectores 0's y 1's se los denomina sparse vectors.

Bolsa de Palabras BoW



Otra versión va enumerando la cantidad de apariciones de cada palabra.

Que pasaría si tenemos un millón de frases con mas palabras? ¿Cómo se manejaría el orden?

A esto se le llama bolsa de palabras

Bolsa de Palabras – Orden de las palabras

Una solución es trabajar con **n-gramas** lo que nos permite saber que tan probable es la sentencia conformada por **n-palabras**.

Los **n-gramas** pueden estar conformados por:

unigrama

bigrama

trigrama

ó

n-gramas

hola, como, estas, hoy -> unigramas
como estas -> bigrama
como estas hoy -> trigrama

Los n-gramas suele aportar mas contexto a la información para n mayores, sin embargo requiere un plus al analizar las posibles combinaciones de n-palabras

Bolsa de Palabras – Orden problema de los n gramas

Para disminuir la cantidad de combinaciones se pueden tomar solo determinados **n-gramas.**

¿Con cuales palabras nos quedamos?

¿Las menos repetidas? No, sueles ser palabras de poco aporte o errores

¿Las mas repetidas? Si no son stop words.

Frecuencia del termino (TF)

Pudimos llegar a un numero reducido de **n-gramas** pero sigue siendo grande. Por ende nacieron los TF.

Las maneras de calcular el tf son:

- Clasificación binaria, se encuentra o no se encuentra
- Conteo exhaustivo, apariciones en el documento.
- TF: contar apariciones del termino y dividirlo sale la totalidad de términos del documentos
- Normalización Logarítmica: tomando el log del recuento.

TF- IVF (Inverse Document Frecuency)

Pudimos llegar a un numero reducido de **n-gramas** pero sigue siendo grande. Por ende nacieron los TF.

N: Totalidad de los comentarios

D: Conjunto de todos los comentarios

[$\{d \in D : t \in d\}$**]** con **d** cada documento y **t** el termino en el documento

$$\mathrm{idf}(t, D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

Logaritmo de N (totalidad de comentarios) sobre los comentarios que tengan el termino especifico

TF- IVF (Inverse Document Frecuency)

Con esto, TF-IDF es el producto de TF e IDF

$$\mathbf{tfidf}(t, d, D) = \mathbf{tf}(t, d) \cdot \mathbf{idf}(t, D)$$

```
from sklearn.feature_extraction.text import TfidfVectorizer

import pandas as pd
comentarios = ["esta pelicula es malisima", "esta pelicula no es
malisima", "esta pelicula es buenisima", "malisima", "me gustó", "no me
gustó", "no creo que sea una buena película", "es buenisima"]

tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1,2))


features = tfidf.fit_transform(comentarios)

df = pd.DataFrame(
features.todense(),
columns=tfidf.get_feature_names()
)
print(df)
```

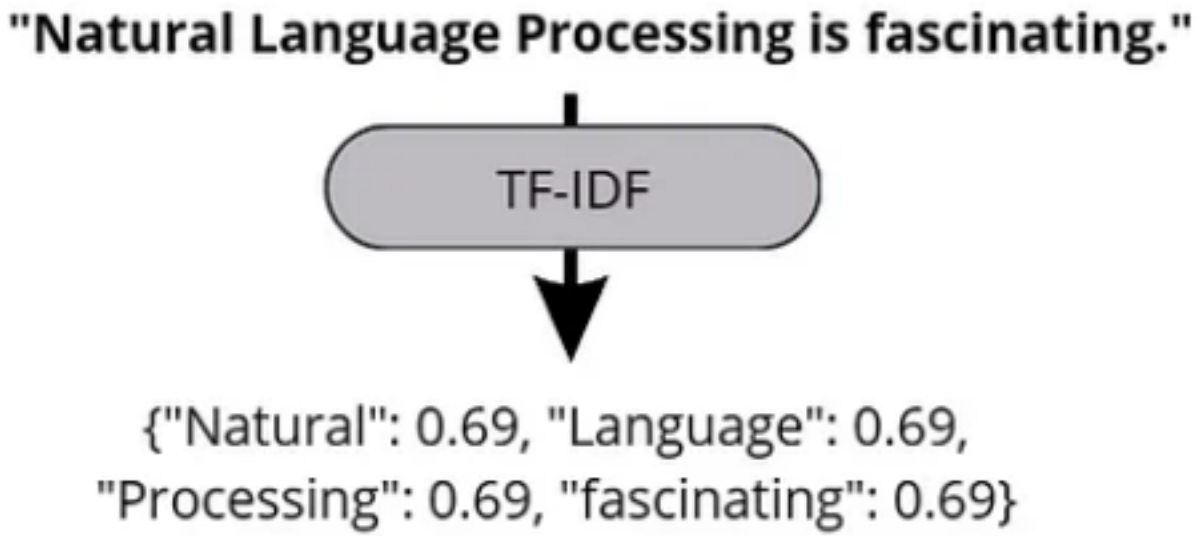
TfidVectorizer: definimos los parámetros minimos y máximos de frecuencia, además de los n-gramas que vamos a analizar (1 y 2)

TF- IVF (Inverse Document Frecuency)

	buenisima	es	es buenisima	es malisima	esta	esta pelicula	\
0	0.000000	0.321127	0.000000	0.424441	0.366257	0.366257	
1	0.000000	0.328779	0.000000	0.434554	0.374985	0.374985	
2	0.415002	0.313986	0.415002	0.000000	0.358112	0.358112	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
7	0.623489	0.471725	0.623489	0.000000	0.000000	0.000000	

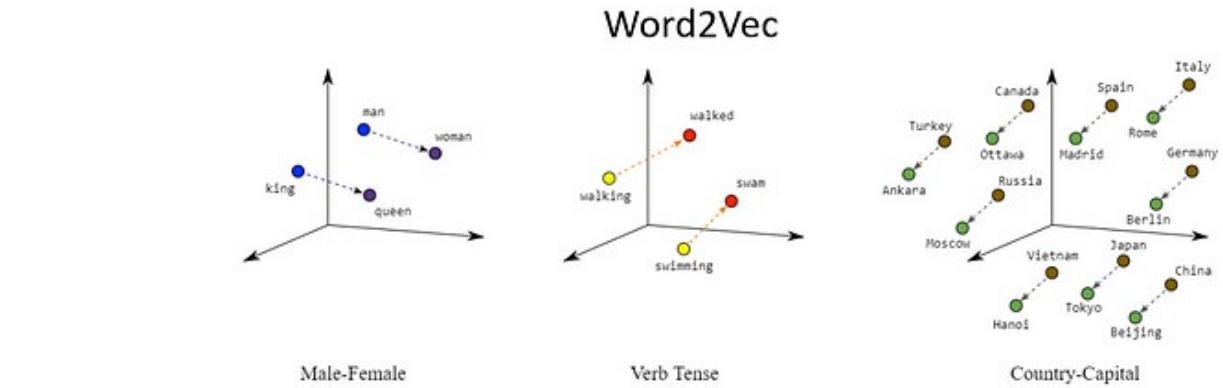
	gustó	malisima	me	me gustó	no	pelicula	pelicula es
0	0.000000	0.366257	0.000000	0.000000	0.000000	0.366257	0.424441
1	0.000000	0.374985	0.000000	0.000000	0.374985	0.374985	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.358112	0.415002
3	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.577350	0.000000	0.577350	0.577350	0.000000	0.000000	0.000000
5	0.516768	0.000000	0.516768	0.516768	0.445928	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

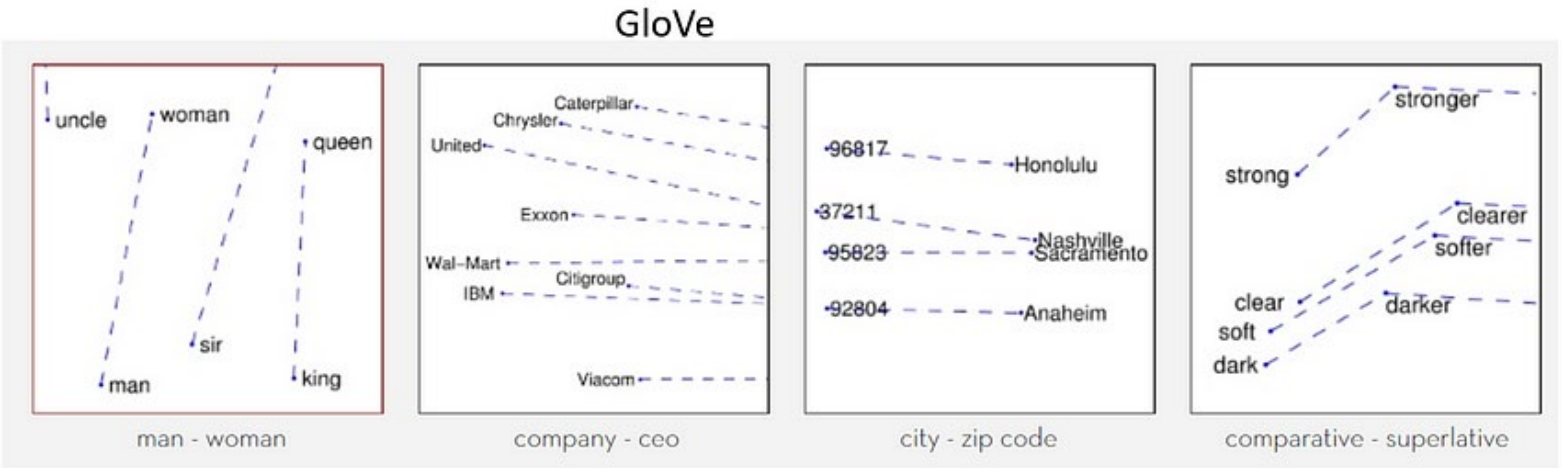
TF-IDF


"Natural Language Processing is fascinating."
TF-IDF
{"Natural": 0.69, "Language": 0.69,
"Processing": 0.69, "fascinating": 0.69}

Word Embeddings

Representa palabras como vectores en un espacio multidimensional.





Word Embeddings

Características:

1. **Representaciones Distribuidas**: Cada palabra se representa como un vector denso de números reales en un espacio de alta dimensión, donde la proximidad entre vectores refleja la similitud semántica entre las palabras.
2. **Captura de Significado**: Los embeddings de palabras capturan el significado semántico de las palabras en función de su contexto de uso. Palabras similares en contexto tienen vectores cercanos en el espacio semántico.
3. **Relaciones Semánticas**: Los embeddings de palabras pueden capturar relaciones semánticas como sinónimos, antónimos y relaciones de analogía (por ejemplo, "rey" está a "hombre" como "reina" está a "mujer").

Métodos de Creación:

1. **Word2Vec**: Desarrollado por Mikolov et al., Word2Vec es un popular algoritmo para entrenar embeddings de palabras utilizando modelos de lenguaje continuo (CBOW). Estos modelos predicen palabras vecinas dada una palabra de entrada.
2. **GloVe** (Global Vectors for Word Representation): Desarrollado por Pennington et al., GloVe utiliza matrices de co-ocurrencia de palabras para aprender embeddings de palabras que capturan la distribución estadística de las palabras en un corpus.
3. **FastText**: Desarrollado por Facebook AI Research, FastText es una extensión de Word2Vec que utiliza subpalabras (n-gramas de caracteres) para representar palabras. Esto permite capturar la información morfológica y de subpalabras de las palabras.

Redes

BoW + red neuronal básica

Clasificador por CNN

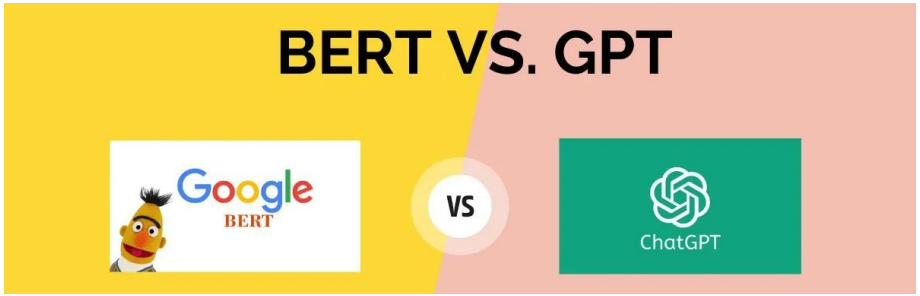
Siguiente clase:

RNN

LSTM

Modelos de lenguaje avanzado Gpt - Bert

GTP



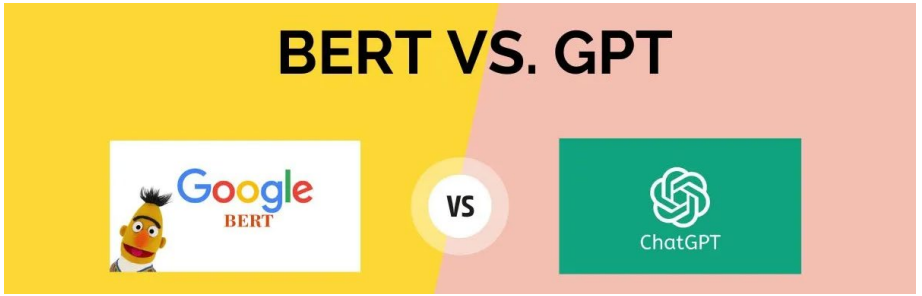
Pertenece a la familia de arquitecturas transformer.

Desarrollado por OpenAI, GPT está entrenado en un vasto corpus de datos de texto diverso, lo que le permite generar texto humano coherente y contextualmente relevante.

Lo que distingue a GPT es su naturaleza autoregresiva, prediciendo la siguiente palabra en una secuencia basada en el contexto precedente.

Este enfoque resulta en una generación de texto fluida y coherente, convirtiendo a GPT en una potencia en tareas como comprensión del lenguaje, completado y generación de texto creativo.

BERT



se destaca como un avance en el PLN al aprovechar la comprensión de contexto bidireccional.

A diferencia de los modelos tradicionales que procesan texto de manera unidireccional, BERT considera tanto el contexto izquierdo como el derecho, mejorando su comprensión de la semántica de las palabras.

Desarrollado por Google, sobresale en tareas que requieren una comprensión profunda de las sutilezas del lenguaje, incluido el análisis de sentimientos, la respuesta a preguntas y la traducción de idiomas.

Su preentrenamiento en conjuntos de datos masivos equipa a BERT para ofrecer un rendimiento incomparable en una amplia gama de tareas relacionadas con el lenguaje.