# *IVÁN FALCÓN MONZÓN*

Video tutorial de la tarea: https://youtu.be/Z1VsHYcNXDI?si=PxIOcNjWb1HItQqB

### INSTALAR GENSIM

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensi
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensi
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->
```

📌 ¿Qué es Gensim?

Gensim es una biblioteca de Python especializada en el procesamiento de texto, aprendizaje automático y modelado de temas. Se utiliza principalmente en tareas de procesamiento de lenguaje natural (NLP) y análisis semántico.

Es especialmente eficiente para trabajar con grandes volúmenes de texto, ya que está diseñada para manejar datos de forma escalable, sin necesidad de cargar todo en memoria.

# ENTRENAR WORD2VEC MODEL (1º PARTE)

Descargar csv

https://www.kaggle.com/datasets/rootuser/worldnews-on-reddit

```
# Importación de librerías necesarias
# Gensim: Librería para modelos de procesamiento de texto
from gensim.models import Word2Vec, keyedvectors

# Pandas: Librería para manipulación de datos en estructuras tipo DataFrame
import pandas as pd

# NLTK: Biblioteca para procesamiento de lenguaje natural
import nltk

# Se importa una librería diferente a la del video
# TreebankWordTokenizer: Tokenizador de NLTK basado en el estándar Penn Treebank
from nltk.tokenize import TreebankWordTokenizer
```

```
# Carga de datos desde un archivo CSV en un DataFrame de Pandas
df = pd.read_csv('reddit_worldnews_start_to_2016-11-22.csv')
```

```
# Muestra las primeras 10 filas del DataFrame
df.head(10)
```

| | time_created | date_created | up_votes | down_votes | title | over_18 | a |
|---|---|---|---|---|---|---|---|
| 0 | 1201232046 | 2008-01-25 | 3 | 0 | Scores killed in Pakistan clashes | False | |
| 1 | 1201232075 | 2008-01-25 | 2 | 0 | Japan resumes refuelling mission | False | |
| 2 | 1201232523 | 2008-01-25 | 3 | 0 | US presses Egypt on Gaza border | False | |
| 3 | 1201233290 | 2008-01-25 | 1 | 0 | Jump-start economy: Give health care to all | False | fa |
| 4 | 1201274720 | 2008-01-25 | 4 | 0 | Council of Europe bashes EU&UN terror | False | mhe |

```
# Extrae los títulos de noticias del DataFrame y los almacena en una variable
newsTiles = df['title'].values
```

```
# Mostrar el contenido de la variable newsTiles
newsTiles
```

```
array(['Scores killed in Pakistan clashes',
       'Japan resumes refuelling mission',
       'US presses Egypt on Gaza border', ...,
       'Professor receives Arab Researchers Award',
       'Nigel Farage attacks response to Trump ambassador tweet',
       'Palestinian wielding knife shot dead in West Bank: Israel police'],
      dtype=object)
```

```
# Inicializa el tokenizador de NLTK basado en el estándar Penn Treebank
tokenizer = TreebankWordTokenizer()

# Tokeniza cada título de noticia en newsTiles usando el tokenizador
newsVec = [tokenizer.tokenize(title) for title in newsTiles]
```

```
# Muestra los títulos tokenizados
newsVec
```

```
 'enforcer'],
['Settlers', 'vow', 'revenge', 'over', 'Jerusalem', 'massacre'],
['Musharraf',
 'Opponents',
 'to',
 'Form',
 'Coalition',
 '(',
 'Musharraf',
 ',',
 'sore',
 'loser',
 'much',
 '?',
 ')'],
['Woman',
 'Earns',
 'Silver',
 'Star',
 'in',
 'Afghan',
 'War',
 '(',
 'only',
 '2nd',
 'woman',
 'to',
 'receive',
 'honor',
 'since',
 'WWII',
 ')'],
['Sky',
 'Train',
 'to',
 'Tibet',
 '...',
 'if',
 'you',
 'can',
 't',
 'free',
 'em',
 ',',
 'visit',
 'em',
 '!'],
...]
```

```
# Entrenamiento del modelo Word2Vec con los títulos tokenizados
model = Word2Vec(newsVec, min_count=1, vector_size=32)
```

```
# Encuentra las palabras más similares a 'man' en el espacio vectorial de Word2Vec
model.wv.most_similar('man')
```

```
[('woman', 0.9733664989471436),
 ('couple', 0.9057114720344543),
 ('girl', 0.8973337411880493),
 ('mother', 0.8929951786994934),
 ('teenager', 0.8903069496154785),
```

```
('boy', 0.8887144923210144),
('doctor', 0.883147656917572),
('family', 0.8660606741905212),
('father', 0.8596243858337402),
('teacher', 0.8555063009262085)]
```

```
# Realiza un cálculo de vectores para encontrar la relación entre 'King', 'man' y 'woman'
vec = model.wv['King'] - model.wv['man'] + model.wv['woman']

# Encuentra las palabras más similares al nuevo vector calculado
model.wv.most_similar([vec])
```

```
[('King', 0.9647098183631897),
 ('king', 0.8357094526290894),
 ('Gandhi', 0.7892224788665771),
 ('Prince', 0.7680248022079468),
 ('prince', 0.7658488154411316),
 ('Abdullah', 0.7640596628189087),
 ('Grand', 0.7507221102714539),
 ('blogger', 0.7480340600013733),
 ('beard’', 0.7424099445343018),
 ('Queen', 0.7418859601020813)]
```

```
# Muestra el vector correspondiente a la palabra 'man' en el modelo Word2Vec
model.wv['man']
```

```
array([ 0.30003816, -0.9363402 ,  3.6655989 , -0.78821963, -5.0071387 ,
       -2.9663901 , -2.0818706 ,  6.251971  , -0.8228237 ,  4.808689  ,
       -5.288639  , -2.2700248 , -1.3681668 ,  1.6236919 , -4.651999  ,
        0.61745894, -0.34006727,  0.2010981 ,  0.39965424, -0.16825916,
       -3.2894707 , -1.42793   , -3.1569164 ,  0.2770847 , -1.8748215 ,
        0.9249649 ,  1.9229184 , -0.0235596 , -4.202868  , -0.02437454,
        0.24892275, -1.3844432 ], dtype=float32)
```

# USAR PRE-ENTRENAMIENTO WORD2VEC MODEL, KEYEDVECTORS (2º PARTE)

Descargar Pretrain Word2vec model

https://drive.usercontent.google.com/download?id=0B7XkCwpI5KDYNlNUTTlSS21pQmM&export=download&authuser=1

```
# Importación de la clase KeyedVectors de gensim para manejar modelos preentrenados
from gensim.models import KeyedVectors  # Importación correcta

# Importación de la librería para interactuar con Google Drive en Google Colab
from google.colab import drive

# Montar Google Drive para acceder a los archivos almacenados en él
drive.mount('/content/drive')

# Especificar la ruta completa del archivo del modelo de Word2Vec (se debe cambiar la ruta según dónde se tenga el archivo)
file_path = '/content/drive/MyDrive/7PRO/GoogleNews-vectors-negative300.bin'
```

```
12
13 # Cargar el modelo preentrenado de Word2Vec desde el archivo binario usando la ruta especificada
14 # El parámetro 'binary=True' indica que el archivo está en formato binario y 'limit=100000' carga solo las primeras 100,000 palabras.
15 model = KeyedVectors.load_word2vec_format(file_path, binary=True, limit=100000)
16
17 # Confirmar que el modelo se ha cargado correctamente
18 print("Modelo cargado correctamente.")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", f
Modelo cargado correctamente.
```

```
1 # Realiza una operación vectorial para obtener una relación semántica entre 'King', 'man' y 'woman'
2 vec = model['King'] - model['man'] + model['woman']
3
4 # Encuentra las palabras más similares al vector calculado
5 model.most_similar([vec])
```

```
[('King', 0.7780153751373291),
 ('Queen', 0.55495285987854),
 ('Princess', 0.464548259973526),
 ('queen', 0.4479384422302246),
 ('Queen_Elizabeth', 0.4418003261089325),
 ('monarch', 0.43155860900878906),
 ('Empress', 0.42811447381973267),
 ('princess', 0.42595720291137695),
 ('Greene', 0.3987760245800183),
 ('Spalding', 0.39462387561798096)]
```

```
1 # Realiza una operación vectorial para obtener una relación semántica entre 'Germany', 'Berlin' y 'Madrid'
2 vec = model['Germany'] - model['Berlin'] + model['Madrid']
3
4 # Encuentra las palabras más similares al vector calculado
5 model.most_similar([vec])
```

```
[('Spain', 0.7713854908943176),
 ('Madrid', 0.754533588886261),
 ('Barcelona', 0.6232754588127136),
 ('Real_Madrid', 0.6106423735618591),
 ('Spaniards', 0.6037396192550659),
 ('Sevilla', 0.5966300368309021),
 ('Atletico_Madrid', 0.5764991641044617),
 ('Barça', 0.568962037563324),
 ('Portugal', 0.568053662776947),
 ('FC_Barcelona', 0.5507020950317383)]
```

```
1 # Realiza una operación vectorial para obtener una relación semántica entre 'Messi', 'football' y 'cricket'
2 vec = model['Messi'] - model['football'] + model['cricket']
3
4 # Encuentra las palabras más similares al vector calculado
5 model.most_similar([vec])
```

```
[('Messi', 0.7753990292549133),
 ('Tendulkar', 0.7388709187507629),
 ('Sehwag', 0.7030156254768372),
 ('Sachin_Tendulkar', 0.6963865160942078),
 ('Dhoni', 0.6875312328338623),
 ('Yuvraj', 0.6863861083984375),
 ('Dravid', 0.6852534413337708),
```

```
('Ponting', 0.6836135387420654),
('Inzamam', 0.6815570592880249),
('Ganguly', 0.6674576997756958)]
```

```
# Realiza una operación vectorial para obtener una relación semántica entre 'Messi', 'football' y 'tennis'
vec = model['Messi'] - model['football'] + model['tennis']

# Encuentra las palabras más similares al vector calculado
model.most_similar([vec])
```

```
[('Messi', 0.7960925102233887),
 ('Lionel_Messi', 0.7120644450187683),
 ('Nadal', 0.6976751685142517),
 ('Del_Potro', 0.6955868005752563),
 ('Xavi', 0.6640554666519165),
 ('Federer', 0.6603957414627075),
 ('Ronaldinho', 0.6550597548484802),
 ('Safin', 0.6450798511505127),
 ('Iniesta', 0.642850935459137),
 ('Wawrinka', 0.638897180557251)]
```

# Repositorios:

Google Colab:
https://colab.research.google.com/drive/1M9rIN34j9dzDmotxVwOcn_70Mrl_KQeu?usp=sharing

Github:
https://github.com/IvanFalconMonzon/TA7_Gensim_Word2Vec_KeyedVectors_IvanFalconMonzon.git