IVÁN FALCÓN MONZÓN

Análisis y Aplicación de Gensim y Word2Vec

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

1. Introducción

¿Qué es Gensim?

Gensim es una biblioteca de **Python** especializada en el p**rocesamiento 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.

Esta diseñada para el modelado de temas y procesamiento de texto en tareas de NLP. Su capacidad para manejar grandes volúmenes de texto sin cargarlos en memoria la hace ideal para entrenar modelos como Word2Vec.

2. Instalación de Gensim

Antes de usar **Gensim**, es necesario instalarlo en el entorno de Google Colab con el siguiente comando:

```
# IVAN FALCON MONZON
!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 gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
```

Si ya está instalado, el sistema confirmará la instalación.

3. Entrenamiento de un modelo Word2Vec desde cero

3.1 Descarga del dataset

Se obtiene un dataset de noticias de Reddit desde Kaggle: https://www.kaggle.com/datasets/rootuser/worldnews-on-reddit

3.2 Carga y preprocesamiento de datos

Cargar/Importar todas las librerías necesarías:

```
# IVAN FALCON MONZON
# 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 vídeo
# TreebankWordTokenizer: Tokenizador de NLTK basado en el estándar Penn Treebank
from nltk.tokenize import TreebankWordTokenizer
```

Los datos se cargan en un DataFrame usando Google drive:

```
from google.colab import drive

# Montar Google Drive
```

```
drive.mount('/content/drive')

# Especificar la ruta del archivo dentro de Google Drive
file_path = '/content/drive/MyDrive/7PRO/reddit_worldnews_start_to_2016-11-22.csv'

# Cargar el dataset en un DataFrame
df = pd.read_csv(file_path)

# Mostrar las primeras filas para verificar la carga
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

	time_created	date_created	up_votes	down_votes	title	over_18	author	subreddit
0	1201232046	2008-01-25	3	0	Scores killed in Pakistan clashes	False	polar	worldnews
1	1201232075	2008-01-25	2	0	Japan resumes refuelling mission	False	polar	worldnews
2	1201232523	2008-01-25	3	0	US presses Egypt on Gaza border	False	polar	worldnews
3	1201233290	2008-01-25	1	0	Jump-start economy: Give health care to all	False	fadi420	worldnews
4	1201274720	2008-01-25	4	0	Council of Europe bashes EU&UN terror blacklist	False	mhermans	worldnews

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

	time_created	date_created	up_votes	down_votes	title	over_18	author	subreddit
0	1201232046	2008-01-25	3	0	Scores killed in Pakistan clashes	False	polar	worldnews
1	1201232075	2008-01-25	2	0	Japan resumes refuelling mission	False	polar	worldnews
2	1201232523	2008-01-25	3	0	US presses Egypt on Gaza border	False	polar	worldnews
3	1201233290	2008-01-25	1	0	Jump-start economy: Give health care to all	False	fadi420	worldnews
4	1201274720	2008-01-25	4	0	Council of Europe bashes EU&UN terror blacklist	False	mhermans	worldnews
5	1201287889	2008-01-25	15	0	Hay presto! Farmer unveils the illegal mock-...	False	Armagedonovich	worldnews
6	1201289438	2008-01-25	5	0	Strikes, Protests and Gridlock at the Poland-U...	False	Clythos	worldnews
7	1201536662	2008-01-28	0	0	The U.N. Mismanagement Program	False	Moldavite	worldnews
8	1201558396	2008-01-28	4	0	Nicolas Sarkozy threatens to sue Ryanair	False	Moldavite	worldnews

Se extraen los títulos de las noticias para su procesamiento posterior:

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

```
# IVAN FALCON MONZON
# 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)
```

3.3 Tokenización de textos

Se usa TreebankWordTokenizer de NLTK para dividir los títulos en palabras:

```
# IVAN FALCON MONZON
# Librería necesaría (cargada anteriormente):
# from nltk.tokenize import TreebankWordTokenizer

# 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]
```

Resultados de **títulos tokenizados**

```
# IVAN FALCON MONZON
```

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

```
 'vet',
 '.',
 'the',
 "'Work',
 'for',
 'food'',
 'placards',
 'go',
 'up'],
['Kosovo', 'Independence', ':', 'One', 'Chronicler', 's', 'Ambivalence'],
['Gaza', 'boy', 'killed', 'in', 'Israeli', 'raid'],
['Jose',
 'Nazario',
 'came',
 'home',
 'from',
 'Iraq',
 'a',
 'hero',
 'for',
 'valor',
 'in',
 'the',
 'Battle',
 'of',
 'Fallujah.',
 'Now',
 'he',
 's',
 'facing',
 'criminal',
 'homicide',
 'charges'],
['Florida',
 ':',
 'Latest',
 ':',
 'Divided',
 'board',
 'approves',
 'teaching',
 'of',
 'evolution',
 'as',
 'theory'],
['University',
 'of',
 'Dallas',
 'School',
 'of',
 'Ministry',
 'to',
 'hold',
 'panel',
 'discussions',
 'on',
 'evolution',
 'and',
 'creationism',
```

3.4 Entrenamiento del modelo Word2Vec

Se entrena un modelo de Word2Vec con los datos procesados:

```
# IVAN FALCON MONZON
# Librería necesaría (cargada anteriormente):
# from gensim.models import Word2Vec

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

3.5 Búsqueda de palabras similares

Una vez entrenado el modelo, se pueden encontrar palabras con significados similares:

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

```
[('woman', 0.9710065722465515),
 ('girl', 0.9115543961524963),
 ('couple', 0.9006036520004272),
 ('boy', 0.8946880102157593),
 ('teenager', 0.8891523480415344),
```

```
('mother', 0.8494493961334229),
('teacher', 0.8462465405464172),
('doctor', 0.8405300974845886),
('herself', 0.8209142684936523),
('father', 0.8170242309570312)]
```

También se pueden realizar operaciones semánticas, como:

```
# 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', 0.9657350182533264),
 ('king', 0.8410685062408447),
 ('prince', 0.7847070097923279),
 ('Grand', 0.7710606455802917),
 ('Prince', 0.7624472975730896),
 ('throne', 0.7550963163375854),
 ('Imam', 0.7473397254943848),
 ('Rushdie', 0.734864354133606),
 ('Queen', 0.7304045557975769),
 ('Carlos', 0.7297850251197815)]
```

Representación vectorial de una palabra:

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

```
array([ 0.06544407, -1.8845025 ,  4.7763805 , -0.88296765, -3.5101197 ,
       -1.7257606 , -0.95023745,  2.8477042 , -0.59723103,  3.884857  ,
       -5.5029306 , -2.107259  , -3.6194372 ,  4.811494  , -0.6432149 ,
       -0.4664334 , -4.9994535 ,  0.50771856, -2.3917987 ,  1.459067  ,
       -0.98585397, -4.374407  , -1.8800361 ,  0.04037134, -1.2837225 ,
        1.8680325 ,  3.8815675 ,  0.03114181, -1.883593  ,  1.3809474 ,
        2.4764261 , -3.0232809 ], dtype=float32)
```

4. Uso de un modelo preentrenado de Word2Vec

4.1 Descarga del modelo preentrenado

Descargar el archivo y ponerlo en Google Drive, pesa 1,5GB.

Descargar Pretrain Word2vec model: https://drive.usercontent.google.com/download?id=0B7XkCwpI5KDYNlNUTTlSS21pQmM&export=download&authuser=1

4.2 Montaje de Google Drive en Colab

Para cargar el modelo almacenado en Drive:

```
# IVAN FALCON MONZON
# 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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

4.3 Carga del modelo preentrenado

Se utiliza KeyedVectors cargar el modelo:

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

# 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'

# Cargar el modelo preentrenado de Word2Vec desde el archivo binario usando la ruta especificada
# El parámetro 'binary=True' indica que el archivo está en formato binario y 'limit=100000' carga solo las primeras 100,000 palabras.
model = KeyedVectors.load_word2vec_format(file_path, binary=True, limit=100000)
```

Se puede verificar la carga con:

```
# IVAN FALCON MONZON
# Confirmar que el modelo se ha cargado correctamente
print("Modelo cargado correctamente.")
```

```
Modelo cargado correctamente.
```

4.4 Operaciones vectoriales en el espacio semántico

Se pueden realizar comparaciones semánticas similares a las realizadas con el modelo entrenado:

```
# IVAN FALCON MONZON
# Realiza una operación vectorial para obtener una relación semántica entre 'King', 'man' y 'woman'
vec = model['King'] - model['man'] + model['woman']

# Encuentra las palabras más similares al vector calculado
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', 0.39877602458000183),
 ('Spalding', 0.39462387561798096)]
```

4.5 Ejemplos de comparaciones

Ejemplo, comparando países y capitales:

```
# IVAN FALCON MONZON
# Realiza una operación vectorial para obtener una relación semántica entre 'Germany', 'Berlin' y 'Madrid'
vec = model['Germany'] - model['Berlin'] + model['Madrid']

# Encuentra las palabras más similares al vector calculado
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)]
```

Otro ejemplo, comparando fútbol y cricket:

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

# Encuentra las palabras más similares al vector calculado
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)]
```

Otro ejemplo, comparando fútbol y tennis:

```
# IVAN FALCON MONZON
# 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)]
```

5. Referencias y enlaces

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

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