Anexo 7

Proyecto 7: Creación de Words Embedding Nivel 3

Mg. Luis Felipe Bustamante Narváez

Word2Vec

Es un modelo que se utiliza para aprender representaciones vectoriales de palabras. Estas representaciones pueden capturar muchas propiedades lingüísticas de las palabras, como su significado semántico, gramatical y hasta contextual.

Para este tercer ejemplo, usaremos 100 textos aleatorios, desde 20.000 caracteres hasta más de 1.500.000 caracteres, los cuales suman para el entrenamiento un total de ---- caracteres.

Librerías

In [...
```
!pip install pypdf2
```

```
Requirement already satisfied: pypdf2 in c:\users\luis_\anaconda3\envs\notebook\lib\site-pack
ages (3.0.1)
```

In [...
```
!pip install tqdm
```

```
Requirement already satisfied: tqdm in c:\users\luis_\anaconda3\envs\notebook\lib\site-packag
es (4.67.1)
Requirement already satisfied: colorama in c:\users\luis_\anaconda3\envs\notebook\lib\site-pa
ckages (from tqdm) (0.4.6)
```

In [...
```
import string
from gensim.models import Word2Vec
import PyPDF2
import os
from tqdm import tqdm
```

Cargamos el documento

In [...
```
def extraer_texto_desde_pdf(ruta_archivo):
    with open(ruta_archivo, 'rb') as archivo:
        lector = PyPDF2.PdfReader(archivo)
        texto = ''
        for pagina in range(len(lector.pages)):
            texto += lector.pages[pagina].extract_text()
        return texto
```

In [...
```
ruta_carpeta = 'Entrenamiento_Word2Vec/textos'
```

Guardamos todos los textos en una lista

In [...
```
todos_los_textos = []
for archivo in tqdm(os.listdir(ruta_carpeta)):
    if archivo.endswith('.pdf'):
```

```
        ruta_completa = os.path.join(ruta_carpeta, archivo)
        try:
            documento = extraer_texto_desde_pdf(ruta_completa)
            todos_los_textos.append(documento)
        except Exception as e:
            print(f'Error al procesar {archivo}: {e}')
```

```
100%|██████████| 100/100 [12:46<00:00,  7.66s/it]
```

In [...
```
len(todos_los_textos)
```

Out[...
```
100
```

Procesamiento de datos

El objetivo del procesamiento es convertir el documento en una lista de frases, y cada frase en una lista de palabras, eliminando signos de puntuación y convirtiendo todo a minúsculas.

In [...
```
# Dividimos el documento en frases usando la coma como separador
frases_totales = []
caracteres = 0

for documento in todos_los_textos:
    caracteres = caracteres + len(documento)
    frases = documento.split(',')
    frases_totales.extend(frases)
```

In [...
```
# Mostramos el número de caracteres totales
print(f'Número de caracteres: {caracteres}')
```

```
Número de caracteres: 240745048
```

In [...
```
# Mostramos el número de oraciones totales
print(f'El número de frases totales es de: {len(frases_totales)}')
```

```
El número de frases totales es de: 1980300
```

In [...
```
# Mostramos un ejemplo
frases_totales[500]
```

Out[...
```
'\ninfluyendo en el desarrollo de la humanidad en múltiples aspectos. A medida que avanzamos en el\nconocimiento'
```

In [...
```
# Mostramos un ejemplo
frases_totales[3000]
```

Out[...
```
' nuevas perspectivas emergen'
```

In [...
```
# Limpiamos las frases
frases_limpias = []
for frase in frases_totales:
    #Eliminamos la puntuación y dividimos por espacios
    tokens = frase.translate(str.maketrans('','',string.punctuation)).split()
    #print(tokens)  #para mostrar qué ha hecho hasta aquí
    #Convertimos a minúsculas
    tokens = [word.lower() for word in tokens]
    #print(tokens)  #para mostrar qué ha hecho hasta aquí
    if tokens:
        frases_limpias.append(tokens)
```

In [...
```
# Mostramos los resultados
frases_limpias[500]
```

Out[…
```
['influyendo',
 'en',
 'el',
 'desarrollo',
 'de',
 'la',
 'humanidad',
 'en',
 'múltiples',
 'aspectos',
 'a',
 'medida',
 'que',
 'avanzamos',
 'en',
 'el',
 'conocimiento']
```

CPU disponibles en mi PC

En este apartado, observaremos la cantidad de núcleos de procesamiento tiene nuestro computador para el trabajo en NPL. Como este modelo requiere de más gasto computacional, es bueno identificar este dato, para ser eficientes en el entrenamiento, y evitar relentizar el equipo u otros procesos en paralelo.

In […
```python
def numero_de_cpus():
    return os.cpu_count()

print(f'Mi equipo tiene {numero_de_cpus()} CPU´s')
```

```
Mi equipo tiene 8 CPU´s
```

Entrenamiento del modelo Word2Vec

In […
```python
model = Word2Vec(sentences=frases_limpias,
                 vector_size=500,
                 window=5,
                 min_count=1,
                 workers=6)
```

Explicación:

- sentences: Es la lista de palabras que vamos a vectorizar
- vector_size: Es el tamaño de dimensiones que le daremos al vector
- window: Son la cantidad de palabras por encima y por debajo que le darán contexto
- min_count: La aparición mínima de una palabra para tenerla en cuenta en el entrenamiento
- workers: Cantidad de núcleo de procesador que vamos a invertir en el entrenamiento

Pruebas

In […
```python
# Verificamos el vector para alguna palabra
vector = model.wv['ciencia']
vector
```

In […
```python
# Mostramos las palabras cercanas
palabras_cercanas = model.wv.most_similar('ciencia', topn=10)
```

```
palabras_cercanas
# Es probable que la similitud falle por tener tan pocas palabras en el texto
```

Out[…
```
[('principios', 0.3798693120479584),
 ('tecnología', 0.36144816875457764),
 ('medicina', 0.2695100009441376),
 ('biografía', 0.2573043406009674),
 ('múltiples', 0.2501693665981293),
 ('múltiplesdimensiones', 0.2432234287261963),
 ('economía', 0.24100026488304138),
 ('relevantebiografía', 0.22121661901474),
 ('vidas', 0.20030616223812103),
 ('han', 0.19419683516025543)]
```

In […
```
# Mostramos las palabras cercanas
palabras_cercanas = model.wv.most_similar('sistemas', topn=10)
palabras_cercanas
# Es probable que la similitud falle por tener tan pocas palabras en el texto
```

Out[…
```
[('enseñanza', 0.8446722030639648),
 ('educación', 0.4766235649585724),
 ('crisis', 0.4423055946826935),
 ('financieras', 0.4304288625717163),
 ('cultural', 0.37052440643310547),
 ('videojuegos', 0.35479822754859924),
 ('actualidad', 0.29080677032470703),
 ('gastronomía', 0.25270408391952515),
 ('fundamentales', 0.2261238843202591),
 ('aportes', 0.2239352911710739)]
```

Guardar modelo

In […
```
model.save('Entrenamiento_Word2Vec/100textos.model')
```

Cargar el modelo

In […
```
modelo_cargado = Word2Vec.load('Entrenamiento_Word2Vec/100textos.model')
```

In […
```
# Probamos con el modelo caragado
palabras_cercanas2 = modelo_cargado.wv.most_similar('importancia', topn=10)
palabras_cercanas2
```

Out[…
```
[('emocional', 0.6839113831520081),
 ('industria', 0.29906395077705383),
 ('cultura', 0.2832416296005249),
 ('másimportantes', 0.24401310086250305),
 ('expertos', 0.2300969511270523),
 ('energía', 0.22232283651828766),
 ('géneros', 0.22123292088508606),
 ('importantes', 0.20623861253261566),
 ('nuclear', 0.19350115954875946),
 ('descubrimientos', 0.18687954545021057)]
```

Guardar Embedido

Existen dos maneras, usando .txt sin binarios, y usando .bin con binarios.

In [...
```
model.wv.save_word2vec_format('Entrenamiento_Word2Vec/100textos_emb.txt', binary=False)
model.wv.save_word2vec_format('Entrenamiento_Word2Vec/100textos_emb.bin', binary=True)
```

Cargar Embedidos

Si se carga el .txt, se usa sin binarios; si se carga el .bin, se usa con binarios

In [...
```
from gensim.models import KeyedVectors
embedding_cargado_txt = KeyedVectors.load_word2vec_format(
    'Entrenamiento_Word2Vec/100textos_emb.txt', binary=False)
```

In [...
```
embedding_cargado_bin = KeyedVectors.load_word2vec_format(
    'Entrenamiento_Word2Vec/100textos_emb.bin', binary=True)
```

In [...
```
# Probamos
embedding_cargado_txt
```

Out[...
```
<gensim.models.keyedvectors.KeyedVectors at 0x1d45dbc0a70>
```

In [...
```
# Probamos
embedding_cargado_bin
```

Out[...
```
<gensim.models.keyedvectors.KeyedVectors at 0x1d444aa0710>
```

Analogías

In [...
```
def analogics(v1, v2, v3):
    simil = embedding_cargado_bin.most_similar(positive=[v1,v3],
                                               negative=[v2]
                                              )
    print(f'{v1} es a {v2}, como {simil[0][0]} es a {v3}')
```

In [...
```
analogics('científico', 'ciencia', 'cultura')
```

```
científico es a ciencia, como pensamiento es a cultura
```

In [...
```
analogics('científico', 'ciencia', 'nuclear')
```

```
científico es a ciencia, como energía es a nuclear
```

In [...
```
analogics('científico', 'ciencia', 'imperio')
```

```
científico es a ciencia, como romano es a imperio
```

Conclusiones

Utilizando 100 textos con temáticas aleatorias, se puede observar que las predicciones en las analogías son mucho más reales; chatGPT3 utilizó en su primer entrenamiento 570G en textos, libros y artículos, nuestro entrenamiento utilizó tan solo 82.8M y aún así, encontramos mucha coherencia a la hora de probar la similitud. ¿Qué pasaría si usaramos por lo menos 1.000.000 de textos?

Mg. Luis Felipe Bustamante Narváez