Unidad: Procesamiento de Lenguaje Natural Moderno (NLP)

Desarrollo con Python + Jupyter Notebook (Kaggle /Hugging Face/ Colab)

Ejercicio 1: Entrenamiento de Word2Vec desde cero

Objetivo: Aprender a construir un modelo de Word Embeddings con Word2Vec.

- Herramientas: gensim, nltk
- Dataset: Corpus de reseñas de películas (nltk.corpus.movie_reviews) from nltk.corpus import movie_reviews from nltk.tokenize import word_tokenize from gensim.models import Word2Vec import nltk nltk.download('movie_reviews') nltk.download('punkt') sentences = [word_tokenize(movie_reviews.raw(fileid)) for fileid in movie_reviews.fileids()] model = Word2Vec(sentences, vector_size=100, window=5, min_count=2, workers=4)

Ejemplo: palabras similares a "good"

print(model.wv.most_similar("good", topn=5))

Resultado esperado: vector de palabras similares a "good", como "great", "nice", etc.

Ejercicio 1: Entrenamiento de Word2Vec desde cero

Preguntas:

1. ¿Qué representa un vector de palabras en Word2Vec?
2. ¿Cuál es la diferencia entre el enfoque CBOW y Skip-Gram?
3. ¿Qué significa que dos palabras tengan vectores "cercanos"?
4. ¿Cómo influye el parámetro window en el entrenamiento?
5. ¿Por qué es necesario hacer tokenización antes de entrenar?

Sugerencias de mejora:

- Agregar visualización de los vectores con TSNE o PCA para mayor comprensión.
- Mencionar si se usa CBOW o Skip-Gram por defecto.

Ejercicio 2: Cargar GloVe y realizar similitud semántica

Objetivo: Utilizar embeddings preentrenados con GloVe para comparar palabras.

- Herramientas: gensim, numpy

- Dataset: glove.6B.100d.txt (disponible en Kaggle) from gensim.models import KeyedVectors

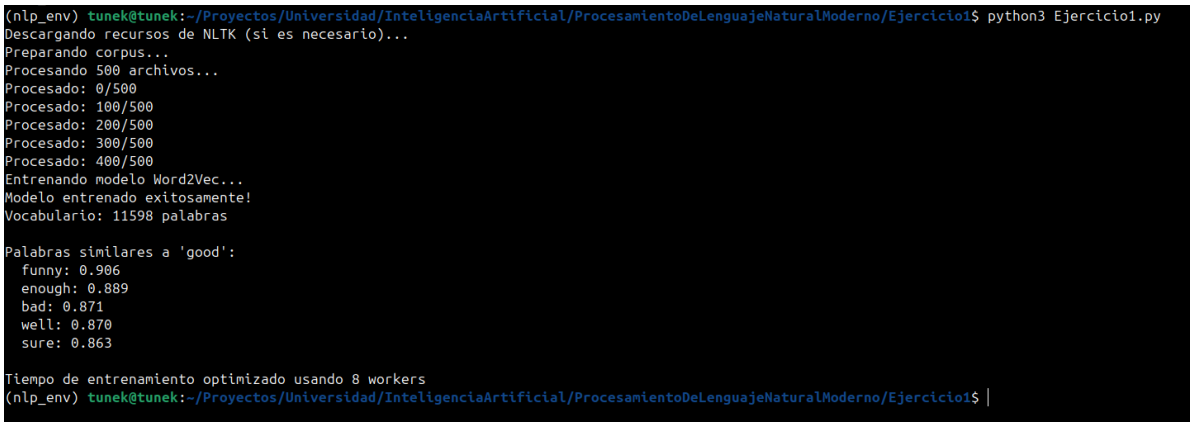
glove_model = KeyedVectors.load_word2vec_format('glove.6B.100d.txt', binary=False, no_header=True)

Similitud entre pares de palabras

glove_model.similarity('king', 'queen') # cercano a 0.8

glove_model.similarity('cat', 'banana') # cercano a 0.2

Resultado esperado: Diferencias semánticas evidentes.



Ejercicio 2: Uso de GloVe y similitud semántica

Preguntas:

1. ¿Cuál es la diferencia entre GloVe y Word2Vec en cuanto a su forma de entrenamiento?
2. ¿Por qué usamos KeyedVectors en este ejercicio?
3. ¿Qué resultados obtuviste al comparar "king" y "queen"? ¿Qué interpretas?
4. ¿Puedes mencionar un caso donde el análisis semántico con GloVe sería útil en la industria?
5. ¿Qué limitaciones tienen los embeddings estáticos como GloVe?

Sugerencias de mejora:

- Mostrar un ejemplo de analogía (king - man + woman ≈ queen) sería didáctico.
- Agregar comentarios sobre la dimensionalidad de los vectores.

Ejercicio 3: Crear embeddings personalizados de un corpus

Objetivo: Generar embeddings personalizados a partir de texto local (dataset propio).

- Herramientas: gensim, nltk
- Dataset: Artículos sobre tecnología (archivo .txt o dataset Kaggle) # Leer corpus

personalizado with open('articulos_tecnologia.txt', 'r', encoding='utf-8') as f:

corpus = f.read()

tokens = [word_tokenize(sent) for sent in nltk.sent_tokenize(corpus)] model_custom =

Word2Vec(tokens, vector_size=50, window=3, min_count=1, workers=2)

model_custom.wv.most_similar("inteligencia")

Resultado esperado: Relación entre términos como "inteligencia", "artificial", "algoritmo".

Ejercicio 3: Embeddings personalizados desde corpus local

Preguntas:

1. ¿Por qué podrías preferir entrenar tus propios embeddings en vez de usar GloVe?
2. ¿Qué características del texto pueden afectar la calidad de los embeddings?
3. ¿Cómo se refleja el dominio del texto en los vectores obtenidos?
4. ¿Qué cambios harías para mejorar la calidad de tus embeddings?
5. ¿Qué usos prácticos tendría este modelo dentro de una empresa?

Sugerencias de mejora:

- Añadir análisis de frecuencia de palabras del corpus como preprocesamiento.
- Comentar la limpieza del texto antes de tokenizar.

```
(nlp_env) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial/ProcesamientoDeLenguajeNaturalModerno/Ejercicio3$ python3 Ejercicio3.py
Verificando recursos de NLTK...
Cargando dataset...
Dataset cargado: 279577 artículos
Procesando 500 artículos...
Procesado: 0/500
Procesado: 100/500
Procesado: 200/500
Procesado: 300/500
Procesado: 400/500
Tokenizando corpus...
Corpus preparado: 100 oraciones
Entrenando modelo con 8 workers...
Modelo entrenado exitosamente!
Vocabulario: 4220 palabras

Palabras similares a 'intelligence':
  artificial: 0.896
  electricity: 0.859
  mastery: 0.857

Palabras similares a 'artificial':
  continues: 0.898
  intelligence: 0.896
  electricity: 0.867

Palabras similares a 'algorithm':
  proprietary: 0.795
  monitored: 0.792
  vicinity: 0.776

Palabras similares a 'machine':
  validation: 0.862
  intuition: 0.858
  abstractions: 0.853

Palabras similares a 'learning':
  machine: 0.806
  predictive: 0.783
  darpa: 0.776

Palabras similares a 'data':
  motivation: 0.647
```

```
Palabras similares a 'data':
  motivation: 0.647
  sas: 0.646
  server: 0.641

Muestra del vocabulario entrenado:
['the', 'and', 'that', 'this', 'for', 'will', 'with', 'are', 'can', 'from']
(nlp_env) tunek@tunek:~/Proyectos/Universidad/InteligenciaArtificial/ProcesamientoDeLenguajeNaturalModerno/Ejercicio3$
```

Ejercicio 4: Clasificación de texto con BERT (Hugging Face)

Objetivo: Clasificar reseñas de películas usando bert-base-uncased.

- Herramientas: transformers, datasets, sklearn
- Dataset: IMDb (disponible en datasets)

from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments from datasets import load_dataset dataset = load_dataset("imdb") tokenizer = BertTokenizer.from_pretrained("bert-base-uncased") def tokenize_fn(example):

return tokenizer(example["text"], padding="max_length", truncation=True)

encoded = dataset.map(tokenize_fn, batched=True) model = BertForSequenceClassification.from_pretrained("bert-base-uncased")

training_args = TrainingArguments(

```
  output_dir="./results", per_device_train_batch_size=8, num_train_epochs=1,
evaluation_strategy="epoch"

)

trainer = Trainer(

  model=model, args=training_args, train_dataset=encoded["train"].select(range(2000)),
eval_dataset=encoded["test"].select(range(500))

)

trainer.train()
```

Resultado esperado: Modelo entrenado capaz de predecir sentimiento positivo/negativo. **Ejercicio**

4: Clasificación de texto con BERT

Preguntas:

1. ¿Cuál es el propósito del proceso de fine-tuning en BERT?
2. ¿Qué diferencias encontraste entre entrenar con un subconjunto pequeño vs. el dataset completo?
3. ¿Qué hace el tokenizer en el pipeline de Hugging Face?
4. ¿Qué métrica usarías para evaluar este modelo?
5. ¿Por qué es más eficaz BERT que un modelo tradicional como Naive Bayes para clasificación de texto?

Sugerencias de mejora:

- Agregar impresión de métricas (accuracy, f1, etc.) luego del trainer.evaluate().
- Ofrecer opción con pipeline() para alumnos con menos experiencia.

Ejercicio 5: Resumen automático de texto con BART

Objetivo: Generar resúmenes de texto con un modelo preentrenado.

- Herramientas: transformers, pipeline
- Texto libre o noticias

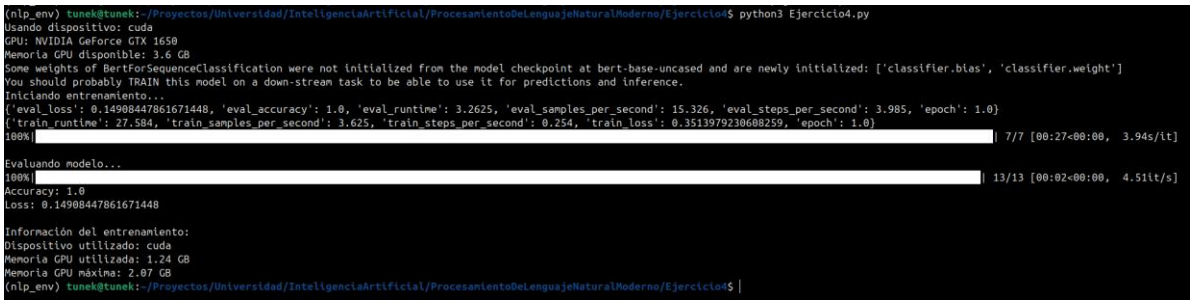
```
from transformers import pipeline


summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
```

text = """La inteligencia artificial está transformando múltiples industrias... (texto largo)"""

summary = summarizer(text, max_length=50, min_length=25, do_sample=False)

print(summary[0]['summary_text'])

Resultado esperado: Un resumen claro y conciso del texto original.



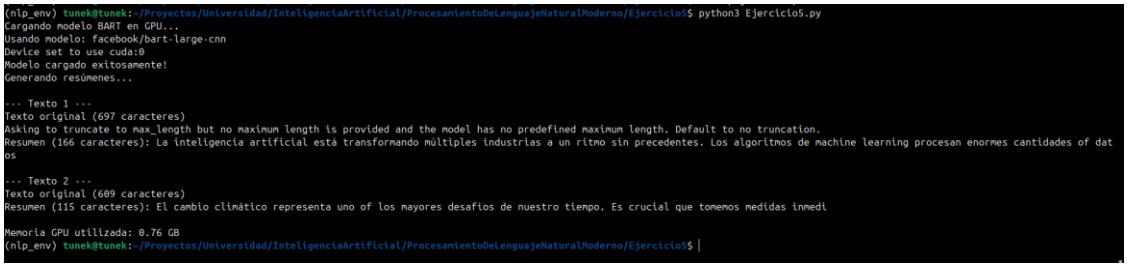
Ejercicio 5: Resumen automático de texto

Preguntas:

1. ¿Cómo se diferencia el resumen extractivo del resumen abstractivo?
2. ¿Por qué usamos facebook/bart-large-cnn para esta tarea?
3. ¿Qué limitaciones encontraste en los resúmenes generados?
4. ¿Cómo podrías ajustar el modelo para resúmenes más cortos o más largos?
5. ¿En qué aplicaciones reales sería útil esta técnica?

Sugerencias de mejora:

- Probar también t5-small para ver diferencias entre modelos.
- Mostrar cómo ajustar max_length, min_length, do_sample para distintos objetivos.



Ejercicio 6: Análisis de sentimientos con DistilBERT

Objetivo: Detectar sentimientos usando distilbert-base-uncased-finetuned-sst-2-english from transformers import pipeline

sentiment = pipeline("sentiment-analysis")

sentiment("This new laptop is amazing!")

sentiment("This was the worst customer service ever.")

Resultado esperado: Sentimiento Positivo o Negativo con nivel de confianza.

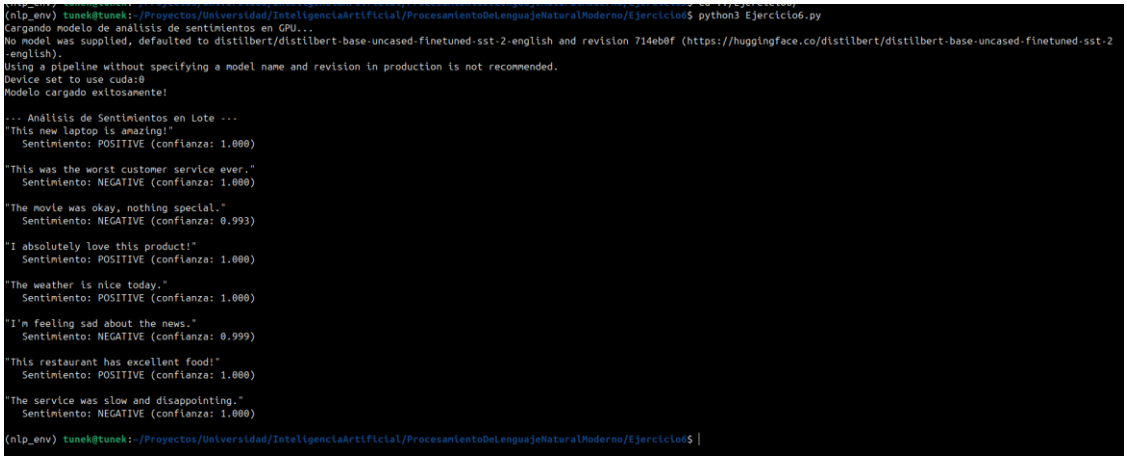
Ejercicio 6: Análisis de sentimientos con DistilBERT

Preguntas:

1. ¿Qué ventajas tiene DistilBERT sobre BERT completo?
2. ¿Qué tipo de tareas reales puedes resolver con análisis de sentimientos?
3. ¿Qué nivel de confianza obtuviste para las frases positivas/negativas?
4. ¿En qué casos podría fallar un modelo de sentimiento?
5. ¿Qué cambios podrías hacer para adaptarlo a un nuevo idioma?

Sugerencias de mejora:

- Agregar visualización del score de sentimiento (por ejemplo, en barra).
- Sugerir evaluación sobre varios ejemplos en lote (batch).



Ejercicio 7: Fine-tuning de BERT en tareas de QA

Objetivo: Ajustar un modelo BERT para responder preguntas sobre contexto.

- Dataset: SQuAD 2.0 (o propio)
- Herramientas: transformers, datasets from transformers import pipeline

qa = pipeline("question-answering", model="distilbert-base-uncased-distilled-squad") qa({

'context': 'La Universidad Técnica de Oruro fue fundada en 1892.',

'question': '¿Cuándo fue fundada la Universidad Técnica de Oruro?'

})

Resultado esperado: "1892"

Ejercicio 7: Fine-tuning para QA con transformers

Preguntas:

1. ¿Qué hace el modelo para identificar la respuesta dentro del contexto?
2. ¿Por qué es útil tener un modelo preentrenado en SQuAD?
3. ¿Qué tan preciso fue el modelo en tus pruebas?
4. ¿Qué desafíos enfrentarías si quisieras entrenar tu propio modelo de QA?
5. ¿Puedes imaginar una aplicación de esta técnica en tu entorno profesional?

Sugerencias de mejora:

- Usar múltiples preguntas sobre un mismo contexto para evaluar comprensión.
- Sugerir prueba con textos propios (p. ej. artículos académicos).

Ejercicio 8: Chatbot básico con Transformers + Gradio

Objetivo: Crear una interfaz conversacional usando un modelo conversacional.

- Herramientas: transformers, gradio
- Modelo: microsoft/DialoGPT-medium import gradio as gr from transformers

import AutoModelForCausalLM, AutoTokenizer import torch tokenizer =

AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium") model =

AutoModelForCausalLM.from_pretrained("microsoft/DialoGPT-medium")

chat_history_ids = None def respond(message, history=[]):

global chat_history_ids input_ids = tokenizer.encode(message +

tokenizer.eos_token, return_tensors='pt')

chat_history_ids = model.generate(input_ids, max_length=1000,
pad_token_id=tokenizer.eos_token_id)

response = tokenizer.decode(chat_history_ids[:, input_ids.shape[-1]:][0],

skip_special_tokens=True) return response

gr.Interface(fn=respond, inputs="text", outputs="text").launch()

Resultado esperado: Interfaz web funcional con respuestas básicas tipo chatbot.

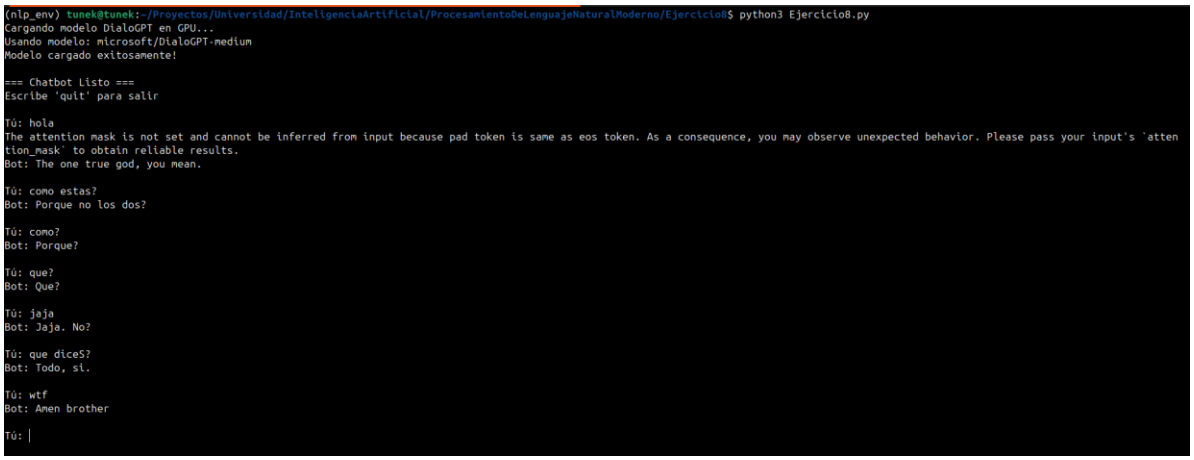
Ejercicio 8: Chatbot con Hugging Face + Gradio

Preguntas:

1. ¿Qué diferencia a un chatbot basado en reglas de uno basado en modelos generativos como DialoGPT?
2. ¿Cómo maneja el modelo el historial de la conversación?
3. ¿Qué problemas encontraste en la coherencia de las respuestas?
4. ¿Qué harías para mejorar la fluidez y precisión del chatbot?
5. ¿Qué otros modelos podrías probar en lugar de DialoGPT?

Sugerencias de mejora:

- Implementar almacenamiento del historial de conversación en history y mostrarlo en UI.
- Comentar los límites de coherencia de modelos pequeños como DialoGP



Ejercicio 9: Proyecto final integrador: Clasificador + Resumen + Sentimiento
Objetivo: Cargar un texto largo, analizar su sentimiento, clasificar su tema y resumirlo.

- Herramientas: transformers, pipeline, gradio (opcional UI) # Carga texto texto = "El

avance de la inteligencia artificial está cambiando el mundo..."

Resumen resumen =

summarizer(texto)[0]['summary_text']

Sentimiento sentimiento = sentiment(texto)[0] # Clasificación temática (usando zero-

shot) classifier = pipeline("zero-shot-classification") result = classifier(texto,

candidate_labels=["tecnología", "política", "salud", "economía"])

print(f"Resumen: {resumen}")

print(f"Sentimiento: {sentimiento}")

print(f"Tema: {result['labels'][0]}")

Resultado esperado: Todo el pipeline NLP funcionando como una miniapp.

Ejercicio 9: Proyecto integrador — Clasificador + Sentimiento + Resumen

Preguntas:

1. Qué tarea resultó más precisa: ¿el resumen, la clasificación o el análisis de sentimiento?
2. ¿Qué tan bien se adaptaron los modelos preentrenados a tu texto personalizado?
3. ¿Cómo integrarías este pipeline en una aplicación web real?
4. ¿Qué parte del pipeline automatizarías o optimizarías con otra herramienta?

5. ¿Qué mejoras podrías hacer si el texto estuviera en otro idioma o jerga regional?

Sugerencias de mejora:

- Dar más detalles sobre el modelo de clasificación zero-shot.
- Proponer extensión del proyecto como **miniAPI web o app de análisis textual**.