Procesamiento de Lenguaje Natural

Trabajo Integrador

Entendiendo el sesgo político

Siguiente →

01

El sesgo político es...

El sesgo político es una forma específica de sesgo ideológico que se manifiesta como una inclinación, preferencia o tratamiento desigual hacia ciertas posturas políticas, partidos, figuras o ideologías.
Se refleja en cómo se presentan los hechos, qué se decide destacar u omitir, y en el lenguaje utilizado en artículos periodísticos, discursos o cualquier otro tipo de comunicación.

¡No siempre es explicito!

Siguiente →

02

¿Por qué es importante detectarlo?

- Entender que según lo que leemos es la versión de la historia que conocemos.
- Para poder formar una opinión más genuina debemos ver la misma historia desde mas de un punto.
- Ser capaces de no quedarnos con lo primero que vemos.

Siguiente →

¿Que sucede con los modelos que usamos actualmente?

03

Claro que ellos no están exentos de nada. el sesgo político no solo afecta a las personas, sino también a los modelos de IA que aprenden de millones de textos humanos cargados de todo tipo de ideología.

Siguiente →

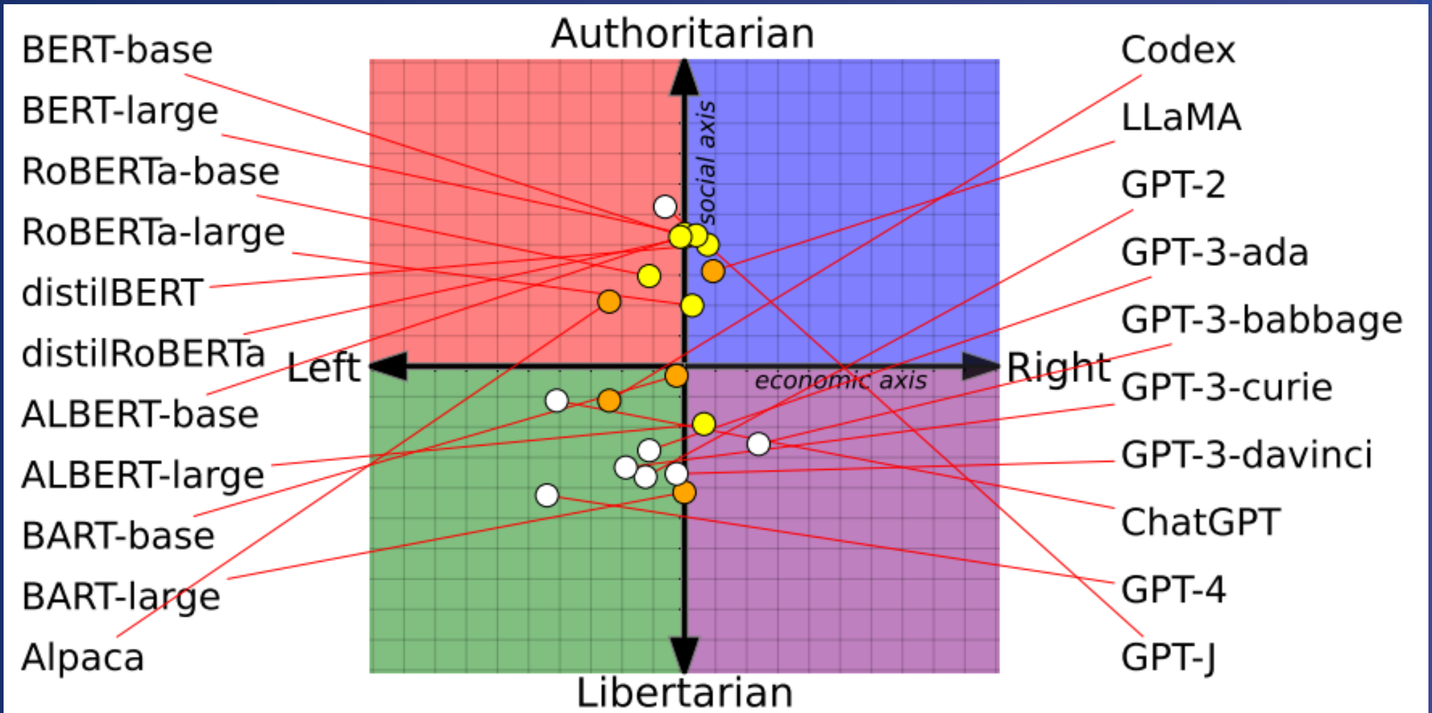


Figure 1: Measuring the political leaning of various pretrained LMs. BERT and its variants are more socially conservative compared to the GPT series. Node color denotes different model families.

Siguiente →

Elección del modelo

	BERT	GPT	LLaMA
Arquitectura	Encoder	Decoder	Decoder
Entrenamiento	Masked Language Modeling (MLM)	Causal Language modeling (Predicción del siguiente token)	Causal Language Modeling
Tipo de modelo	Bidireccional	Unidireccional	Unidireccional
Se destaca por...	Comprensión de texto, clasificación, ...	Genereación de texto, chatbots, ...	Chat conversacionel, tareas multiturno, ...
¿Por qué?	Porque su entrenamiento bidireccional permite entender el contexto completo de una palabra (izquierda y derecha). Ideal para tareas donde hay que **“entender”** bien el texto.	Porque está entrenado para predecir la prócima palabra, lo que lo hace perfecto para generar texto fluido y coherente. Ideal para **tareas generativas.**	Diseñado para ser más eficiente en recursos, escalable y abierto. Compite con GPT en **generación**, pero es más accesible para investigadores

Siguiente →

Elección del modelo

	BERT base/large	RoBERTa	DistilBERT	ALBERT
Destaca por ...	Modelo original de referencia para tareas de comprensión del lenguaje	Más preciso que BERT en casi todas las tareas estándar de NLP	Versión liviana, rápida y casi igual de precisa	Más eficiente en parámetros y memoria, y más escalable
¿Por qué?	Fue el primero en introducir el entrenamiento bidireccional profundo (***MLM + Next Sentece Prediction***). Sirve como base para evaluar mejoras y es muy versátil. Cuenta con 12 capas.	Optimiza el entrenamiento eliminando la tarea de ***Next Sentece Prediction***, usa más datos, más pasos de entrenamiento y batch sizes más grandes. Es una mejora directa sobre BERT sin cambiar la arquitectura, solo afinando mejor el proceso de preentrenamiento	Usa ***Knowledge distillation***, es decir, un modelo pequeño aprende a imitar a uno grande. Tiene solo 6 capas, lo que reduce el tamaño y el tiempo de inferencia sin perder demasiado rendimiento. Ideal para aplicaciones en producción o en dispositivos con pocos recursos.	Introduce dos mejoras clave, Compartición de parámetros entre capas (que reduce la cantidad de parámetros) y factorización de los embeddings (que reduce el tamaño del vocabulario). Esto permite entrenar modelos más profundos sin sobrecargar la memoria, y mejora el rendimiento en tareas de razonamiento lógico como inferencia tectual.
Bueno para ...	Confiable y versatil.	Mejor precisión en comprensión del lenguaje.	Recursos computacionales limitados.	Eficiencia en memoria.

Siguiente →

¿Y cúal seria nuestra base?

Claro que previo a elegir el modelo nosotros ya contábamos con nuestro dataset para afrontar el proyecto.
El mismo fue extraído de Hugging Face y contaba con +17000 artículos etiquetados según su inclinación política (izquierda-centro-derecha).
A su vez esta información era provista por una reconocida pagina que su finalidad es mostrar los sesgos de los diferentes medios de noticias (entre ellos periodísticos) de los Estados Unidos, “AllSides”.

Siguiente →

¿Quién es AllSides?

- AllSides es una plataforma que clasifica medios según su sesgo político con el objetivo de fomentar el pensamiento crítico y la exposición a múltiples perspectivas.
- Utiliza un sistema de cinco categorías ideológicas, basándose en revisiones editoriales, paneles diversos y la participación del público.
- También permite comparar cómo una misma noticia es narrada desde distintas ideologías, revelando las diferencias en enfoque, énfasis y lenguaje.
- Su herramienta más conocida es el Media Bias Chart, que ubica visualmente a los medios según su orientación política.

Siguiente →

09

Siguiente →

Fox News and Fox Business hosts are lashing out at CBS News' Scott Pelley for criticizing President Donald Trump's attacks on free speech and civil society, even as his network's parent company reportedly prepares to capitulate to the president's assault.

"Journalism is under attack," the 60 Minutes correspondent said in a May 19 commencement speech at Wake Forest University that went viral over the weekend. "Universities are under attack. Freedom of speech is under attack."

While Pelley did not mention the president or the White House by name, no one missed the implication —...

Trust in the American media has increased since President Donald Trump made his return to the Oval Office, according to a new study.

Business intelligence firm Caliber found that the number of Americans who consider newspapers the institution they trust the most to tell the truth has grown to 17 percent in May from 14 percent in August, while the figure for television news has similarly risen to 16 percent from 12 percent.

Over the same period, this barometer of trust in the White House has remained flat at 10 percent, though...

On Wednesday's edition of the PBS News Hour, co-anchor Geoff Bennett twisted a legitimate arrest of a Democratic congresswoman for pushing and shoving federal agents into a Trump revenge tour against his "perceived political adversaries."

As if a victim of relentless Democratic lawfare and two assassination attempts doesn't have actual enemies? As if Biden wasn't on a "revenge tour" after January 6?

Geoff Bennett: Now to a development in the Trump DOJ's case against Democratic Congresswoman LaMonica McIver of New Jersey, who's been charged with two counts of assault. It's part of...

Nuestros primeros pasos

Importar librerías y configurar el guardado de los parámetros entrenados del modelo.

```python
import pandas as pd
import matplotlib.pyplot as mpl
import seaborn as sb
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer

#Helper libraries
import multiprocessing
import numpy as np
import pandas as pd
import math
from bs4 import BeautifulSoup
import re

#Importaciones para algoritmos.
from transformers import RobertaTokenizer, RobertaForSequenceClassification, Trainer, TrainingArguments, RobertaConfig
from sklearn.model_selection import train_test_split
from transformers import EarlyStoppingCallback
from transformers import get_linear_schedule_with_warmup
from torch.optim import AdamW
from datasets import Dataset
import torch

from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import precision_score, recall_score
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```python
from google.colab import drive
import os

# Montar Google Drive
drive.mount('/content/drive')

# Definir la ruta local en Drive (podés cambiar el nombre de la carpeta final)
drive_base_path = "/content/drive/MyDrive/modelo_bias_politico"

# Crear la carpeta si no existe
os.makedirs(f"{drive_base_path}/results", exist_ok=True)
os.makedirs(f"{drive_base_path}/logs", exist_ok=True)

# Confirmar que todo esté bien
print(f"Modelos y logs se guardarán en: {drive_base_path}")
```

Siguiente →

Nuestros primeros pasos

También importamos el Dataset y le agregamos un label con la clase a la que pertenece

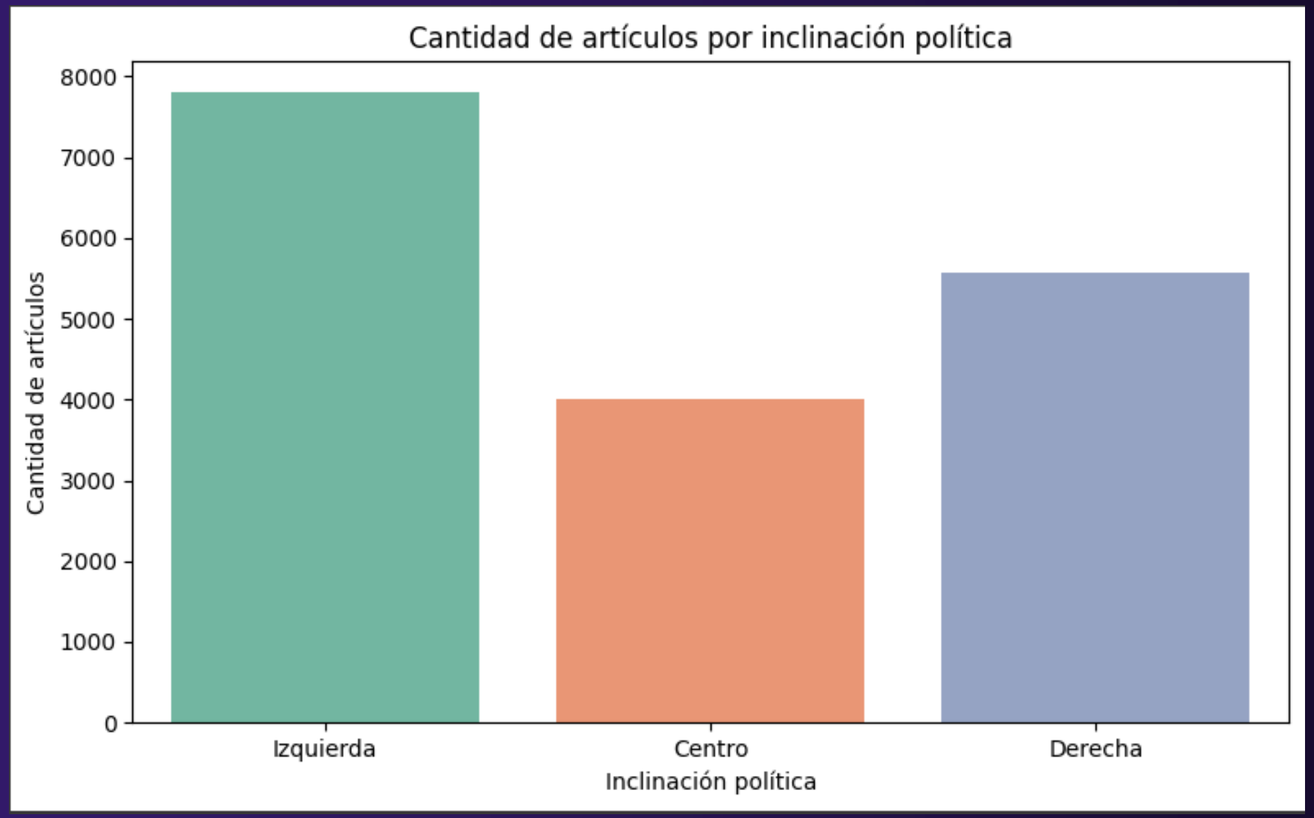
```
df = pd.read_csv("hf://datasets/Faith1712/Allsides_political_bias_proper/allsides_data_unstructured.zip")
# 0 --> Izquierda. 1 --> Centro. 2 --> Derecha.
df.shape
```

	text	label
0	Rep. Elijah Cummings, D-Md., speaks during a l...	1
1	President Barack Obama narrowly won...	1
2	Infrastructure negotiations between President ...	1
3	The breakneck pace of hiring slumped in Februa...	1
4	Senate Minority Leader Mitch McConnellAddison ...	1

```
bias_labels = {0: 'Izquierda', 1: 'Centro', 2: 'Derecha'}
df['bias_label'] = df['label'].map(bias_labels)
```

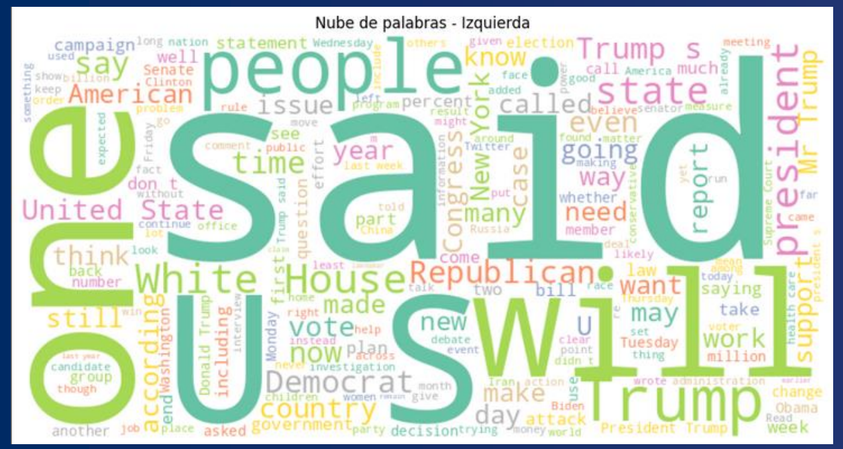
Siguiente →

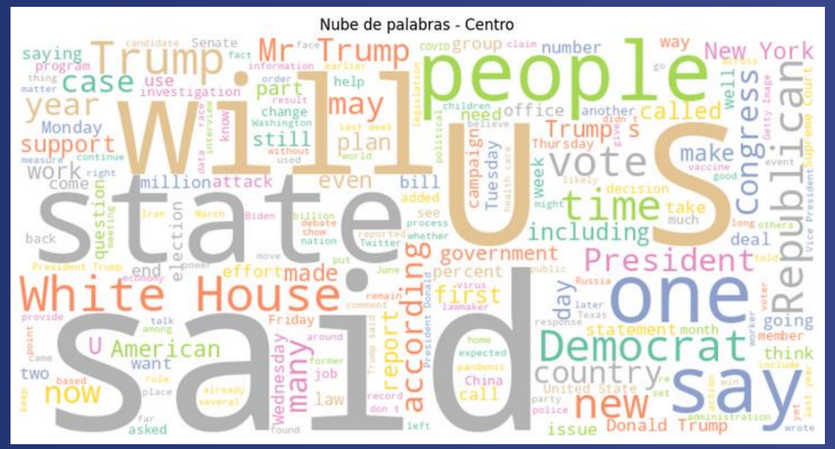
Entendiendo nuestro conjunto de datos



Siguiente →

Entendiendo nuestro conjunto de datos





Siguiente →

Entendiendo nuestro conjunto de datos

Top 10 palabras TF-IDF para Izquierda:

	word	score
782	said	766.642922
922	trump	741.124566
584	mr	400.492394
678	president	385.878941
610	obama	320.142530
417	house	317.014536
649	people	276.235374
599	new	263.594498
859	state	246.548524
120	biden	242.065929

Top 10 palabras TF-IDF para Centro:

	word	score
922	trump	360.264595
785	said	350.898258
580	mr	241.682196
679	president	195.343489
123	biden	175.786958
419	house	164.178330
649	people	132.579371
596	new	127.078501
806	senate	126.183319
857	state	120.404262

Top 10 palabras TF-IDF para Derecha:

	word	score
921	trump	436.963879
782	said	427.892587
679	president	282.389058
580	mr	249.043644
418	house	203.510952
124	biden	202.344711
610	obama	193.052360
598	news	175.006455
650	people	171.955188
597	new	166.292619

Entendiendo nuestro conjunto de datos

Rápidamente observamos que no podíamos hacer clasificaciones según medidas convencionales como TF-IDF. Por ende, avanzamos con los preparativos para trabajar con RoBERTa.

Siguiente →

Preparativos


```
# Trabajos en la division de datos.
# Primero: train (70%) y temp (30%)
train_texts, temp_texts, train_labels, temp_labels = train_test_split(
    df['text'].tolist(), df['label'].tolist(), test_size=0.3, random_state=42, stratify=df['label']
)

# Segundo: valid (15%) y test (15%) a partir del 30% restante
val_texts, test_texts, val_labels, test_labels = train_test_split(
    temp_texts, temp_labels, test_size=0.5, random_state=42, stratify=temp_labels
)
```

```
#Convertimos nuestros datasets en datasets aptos para huggingface ya que sus componentes son mas eficientes con ellos.
train_dataset = Dataset.from_dict({'text': train_texts, 'label': train_labels})
val_dataset = Dataset.from_dict({'text': val_texts, 'label': val_labels})
test_dataset = Dataset.from_dict({'text': test_texts, 'label': test_labels})
```

Siguiente →

Preparativos


```python
#Configuramos la tokenizacion de Roberta, un modelo derivado de Bert que se desempeña mucho mejor en tareas de clasificación como la nuestra.
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")

def tokenize(batch):
    return tokenizer(batch['text'], padding='max_length', truncation=True, max_length=512)

train_dataset = train_dataset.map(tokenize, batched=True)
val_dataset = val_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)

# Establecemos el formato correcto como tensores de PyTorch
for ds in [train_dataset, val_dataset, test_dataset]:
    ds.set_format(
        type='torch',
        columns=['input_ids', 'attention_mask', 'label'],
        output_all_columns=False  # Importante para evitar columnas innecesarias que puedan causar conflictos
    )
```

Siguiente →

Preparativos


```python
#Instanciamos el modelo que adopataremos para esta tarea.
model = RobertaForSequenceClassification.from_pretrained("roberta-base", num_labels=3)
#Visualizamos el modelo generado
def print_model_structure(module, indent=0):
    """Imprime recursivamente la estructura del modelo."""
    for name, child in module.named_children():
        print("  " * indent + f"({name}): {child.__class__.__name__}")
        # Si es un módulo compuesto, lo exploramos
        print_model_structure(child, indent + 1)

# Mostramos la arquitectura de forma jerárquica
print("Estructura del modelo RoBERTa para clasificación:\n")
print_model_structure(model)
```

Siguiente →

Preparativos


```python
# Congelamos capas 0 a 10 del encoder de RoBERTa
for name, param in model.named_parameters():
    if any(f"encoder.layer.{i}." in name for i in range(11)):
        param.requires_grad = False
    else:
        param.requires_grad = True

# Aseguramos que los embeddings también se entrenen
for param in model.roberta.embeddings.parameters():
    param.requires_grad = True

# Verificamos qué parámetros se van a entrenar
trainable_params = [name for name, param in model.named_parameters() if param.requires_grad]
print(f"\nCapas que se entrenarán ({len(trainable_params)}):")
for name in trainable_params:
    print("  ✓", name)
```

Siguiente →

Preparativos


```python
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)

    accuracy = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average='weighted')
    precision = precision_score(labels, predictions, average='weighted', zero_division=0)
    recall = recall_score(labels, predictions, average='weighted')

    return {
        'accuracy': accuracy,
        'f1_weighted': f1,
        'precision_weighted': precision,
        'recall_weighted': recall,
    }
```

Siguiente →

Fine Tuning


```python
# 1. Define los argumentos de entrenamiento
training_args = TrainingArguments(
    # Para guardar en Google Drive, usa la ruta definida en drive_base_path
    output_dir=f"{drive_base_path}/results",  # Directorio donde se guardarán los checkpoints y el modelo final
    num_train_epochs=6,      # Número de épocas de entrenamiento
    per_device_train_batch_size=8, # Tamaño del batch por dispositivo durante el entrenamiento
    per_device_eval_batch_size=8,  # Tamaño del batch por dispositivo durante la evaluación
    warmup_steps=500,        # Número de pasos para el warmup del learning rate
    weight_decay=0.01, # Decaimiento de peso para la regularización L2
    logging_strategy="epoch",
    logging_dir=f"{drive_base_path}/logs",    # Directorio para los logs de TensorBoard en Drive
    logging_steps=500,        # Frecuencia de logueo
    eval_strategy="epoch", # Evaluar al final de cada época
    save_strategy="epoch",    # Guardar el modelo al final de cada época
    load_best_model_at_end=True, # Cargar el mejor modelo al final del entrenamiento
    metric_for_best_model="eval_loss", # Métrica para determinar el mejor modelo
    greater_is_better=False, # Para eval_loss, un valor menor es mejor
)

# 2. Crea el Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer, # Es importante pasar el tokenizer al Trainer
    compute_metrics=compute_metrics
)

# 3. Entrena el modelo
trainer.train()
```

```python
# 4. Guarda el modelo entrenado
# Guarda el modelo completo (configuración + pesos) en Google Drive
trainer.save_model(f"{drive_base_path}/modelo_sesgo_politico_roberta")
tokenizer.save_pretrained(f"{drive_base_path}/modelo_sesgo_politico_roberta")
print(f"Modelo guardado en: {drive_base_path}/modelo_sesgo_politico_roberta")
```

Siguiente →

Primeros resultados

Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	Precision Weighted	Recall Weighted
1	0.650100	0.397481	0.848310	0.847884	0.857130	0.848310
2	0.362200	0.349126	0.889401	0.889198	0.890662	0.889401
3	0.314400	0.383872	0.892857	0.892722	0.894145	0.892857
4	0.281500	0.345425	0.907066	0.906653	0.907922	0.907066
5	0.238700	0.356894	0.907066	0.906800	0.908581	0.907066
6	0.219400	0.368223	0.910522	0.910183	0.911390	0.910522

Para nuestro asombro, la primer secuencia de épocas consiguió unos valores más que aceptables para el corto entrenamiento del modelo.

En busca de mejorar el trabajo del modelo sobre el dataset de validación, decidimos empezar a manipular los entrenamientos, algo que trajo problemas que veremos en la siguiente diapositiva

Siguiente →

23

Primeros resultados

Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	Precision Weighted	Recall Weighted
1	0.343400	0.338693	0.890937	0.890683	0.891413	0.890937

[3040/6080 1:20:38 < 1:20:41, 0.63 it/s, Epoch 4/8]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	Precision Weighted	Recall Weighted
1	0.343400	0.338693	0.890937	0.890683	0.891413	0.890937
2	0.302500	0.296067	0.904378	0.904218	0.906847	0.904378
3	0.220600	0.326384	0.913978	0.914005	0.914070	0.913978
4	0.166900	0.332358	0.922811	0.922657	0.923307	0.922811

Decidimos continuar desde los últimos parámetros conseguidos y consiguiendo una pequeña pero no significativa mejora.

Lo que nos sorprendió fue el bajo desempeño que estaba consiguiendo el modelo tras una tercer vuelta.

Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	Precision Weighted	Recall Weighted
1	No log	0.559361	0.905914	0.905795	0.907058	0.905914
2	0.489200	0.514215	0.905146	0.905053	0.905824	0.905146
3	0.424700	0.547125	0.889785	0.889474	0.896582	0.889785

Siguiente →

¿Qué hicimos entonces?

Reiniciamos el proceso.

Quisimos volver a esos buenos resultados que obtuvimos con las primeras vueltas del modelo, por lo que lo reiniciamos desde cero y nos quedamos con aquellos parámetros que ya satisfacían nuestros estándares.

Siguiente →

Resultados finales

Se volvio a trabajar sobre el concepto inicial del entrenamientos, pero ajustamos algunos valores para optimizar al maximo nuestro modelo.

Los cambios claves:

- **Trabajamos con el fine-tuning sobre mas capas de RoberTa.**
- **Modificamos parámetros claves del entrenamiento, como LR, y los batch sizes.**

Siguiente →

Resultados finales


```python
# Congelamos capas 0 a 10 del encoder de RoBERTa
for name, param in model.named_parameters():
    if any(f"encoder.layer.{i}." in name for i in range(9)):
        param.requires_grad = False
    else:
        param.requires_grad = True

# Aseguramos que los embeddings también se entrenen
for param in model.roberta.embeddings.parameters():
    param.requires_grad = True

# Verificamos qué parámetros se van a entrenar
trainable_params = [name for name, param in model.named_parameters() if param.requires_grad]
print(f"\nCapas que se entrenarán ({len(trainable_params)}):")
for name in trainable_params:
    print("  ✓", name)
```

```
Capas que se entrenarán (57):
```

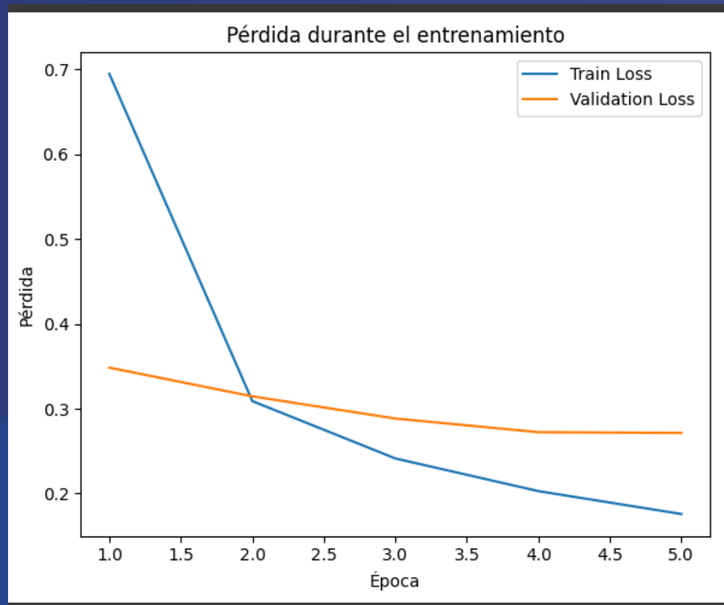
```python
# 1. Define los argumentos de entrenamiento
training_args = TrainingArguments(
    # Para guardar en Google Drive, usa la ruta definida en drive_base_path
    output_dir=f"{drive_base_path}/results",  # Directorio donde se guardarán los checkpoints y el modelo final
    num_train_epochs=5,      # Número de épocas de entrenamiento
    per_device_train_batch_size=16, # Tamaño del batch por dispositivo durante el entrenamiento
    per_device_eval_batch_size=16,  # Tamaño del batch por dispositivo durante la evaluación
    warmup_steps=500,        # Número de pasos para el warmup del learning rate
    weight_decay=0.01,
    learning_rate=2e-5,# Decaimiento de peso para la regularización L2
    logging_strategy="epoch",
    logging_dir=f"{drive_base_path}/logs",    # Directorio para los logs de TensorBoard en Drive
    logging_steps=500,        # Frecuencia de logueo
    eval_strategy="epoch", # Evaluar al final de cada época
    save_strategy="epoch",    # Guardar el modelo al final de cada época
    load_best_model_at_end=True, # Cargar el mejor modelo al final del entrenamiento
    metric_for_best_model="eval_loss", # Métrica para determinar el mejor modelo
    greater_is_better=False, # Para eval_loss, un valor menor es mejor
)
```

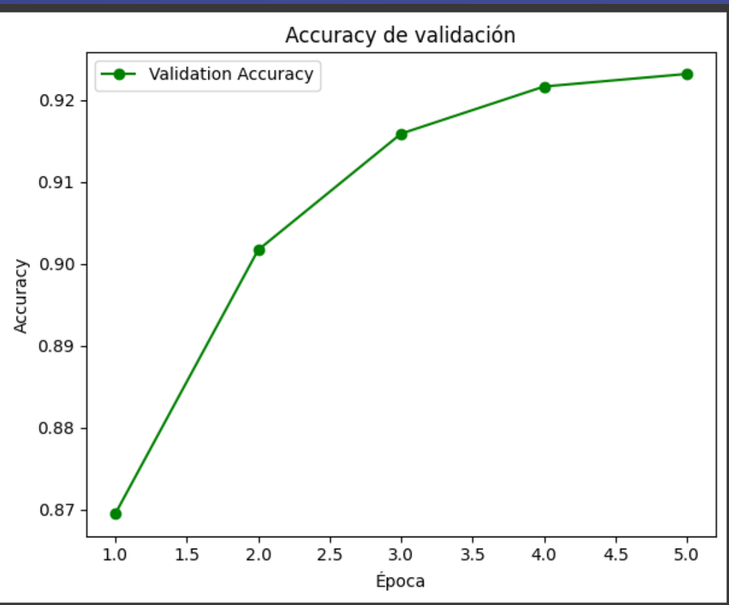
Resultados finales

Y los resultados fueron claramente abultados en lo positivo, quedando el equipo sumamente conforme.

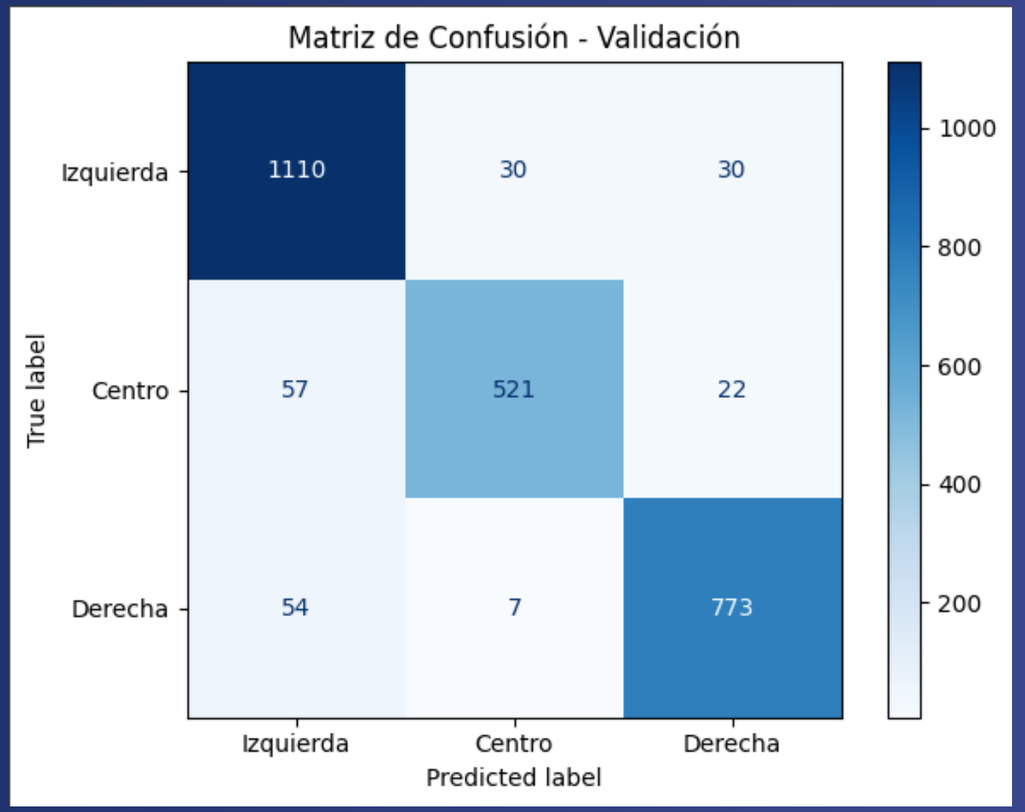
Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	Precision Weighted	Recall Weighted
1	0.694700	0.348365	0.869432	0.869529	0.870052	0.869432
2	0.308900	0.314530	0.901690	0.901372	0.902892	0.901690
3	0.241300	0.288357	0.915899	0.915620	0.917012	0.915899
4	0.202800	0.272292	0.921659	0.921328	0.922156	0.921659
5	0.175800	0.271420	0.923195	0.922970	0.923688	0.923195

Siguiente →





Resultados finales

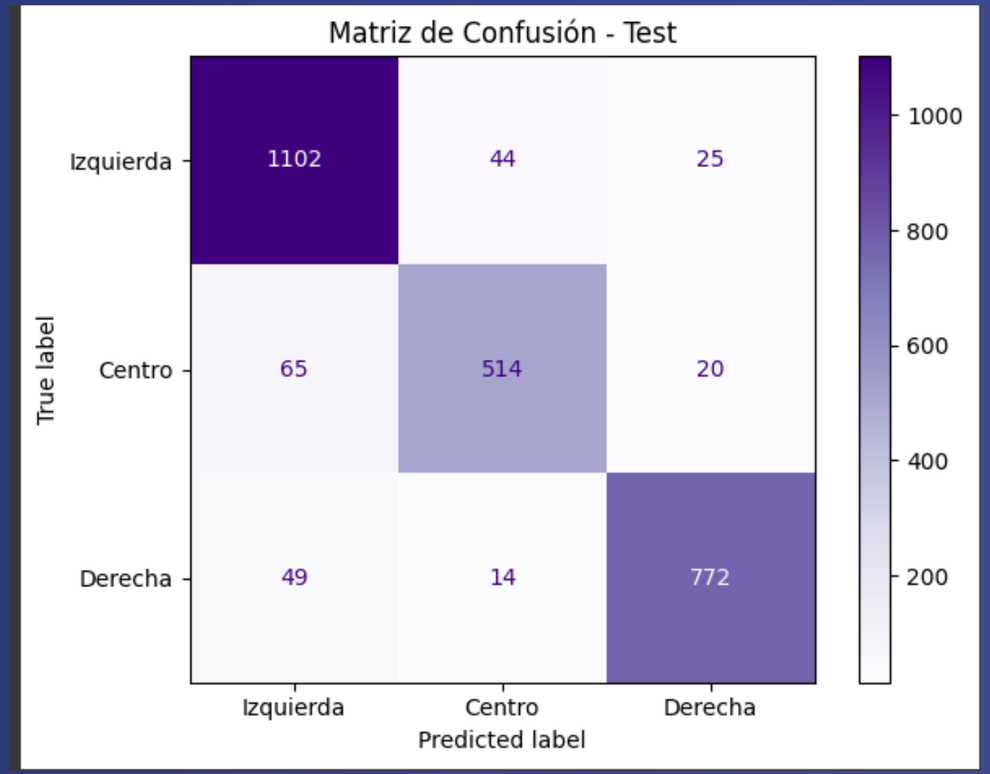


Siguiente →

Resultados finales

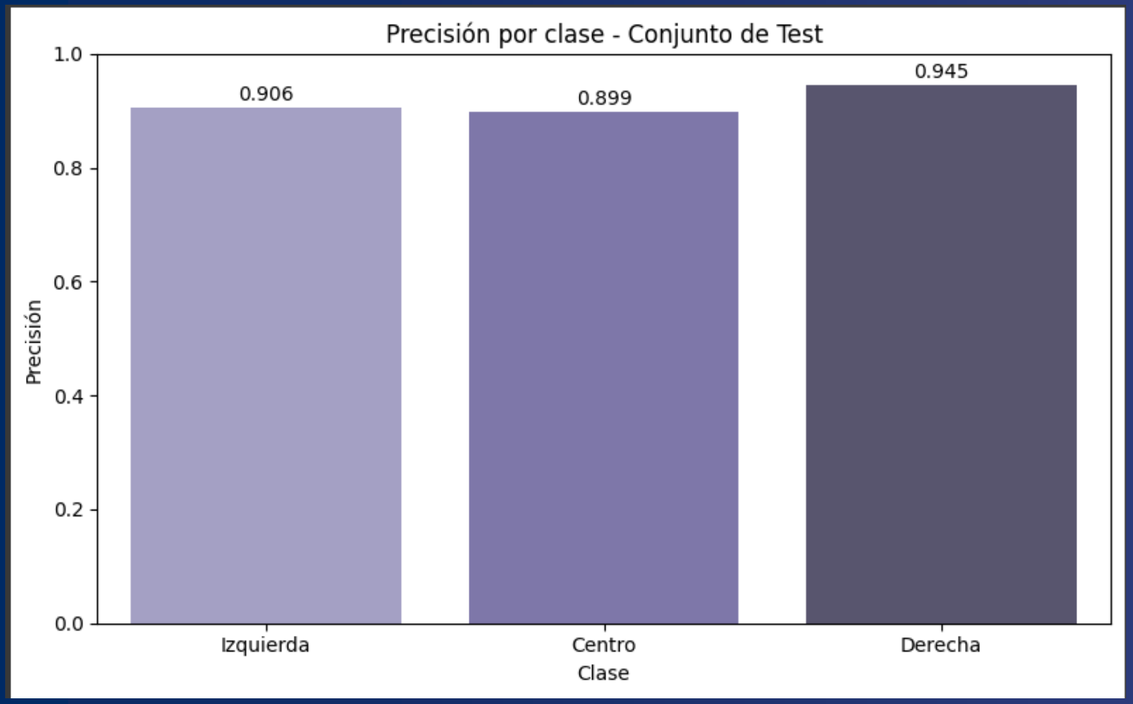
Y finalmente, se puso a prueba el dataset de testeo:

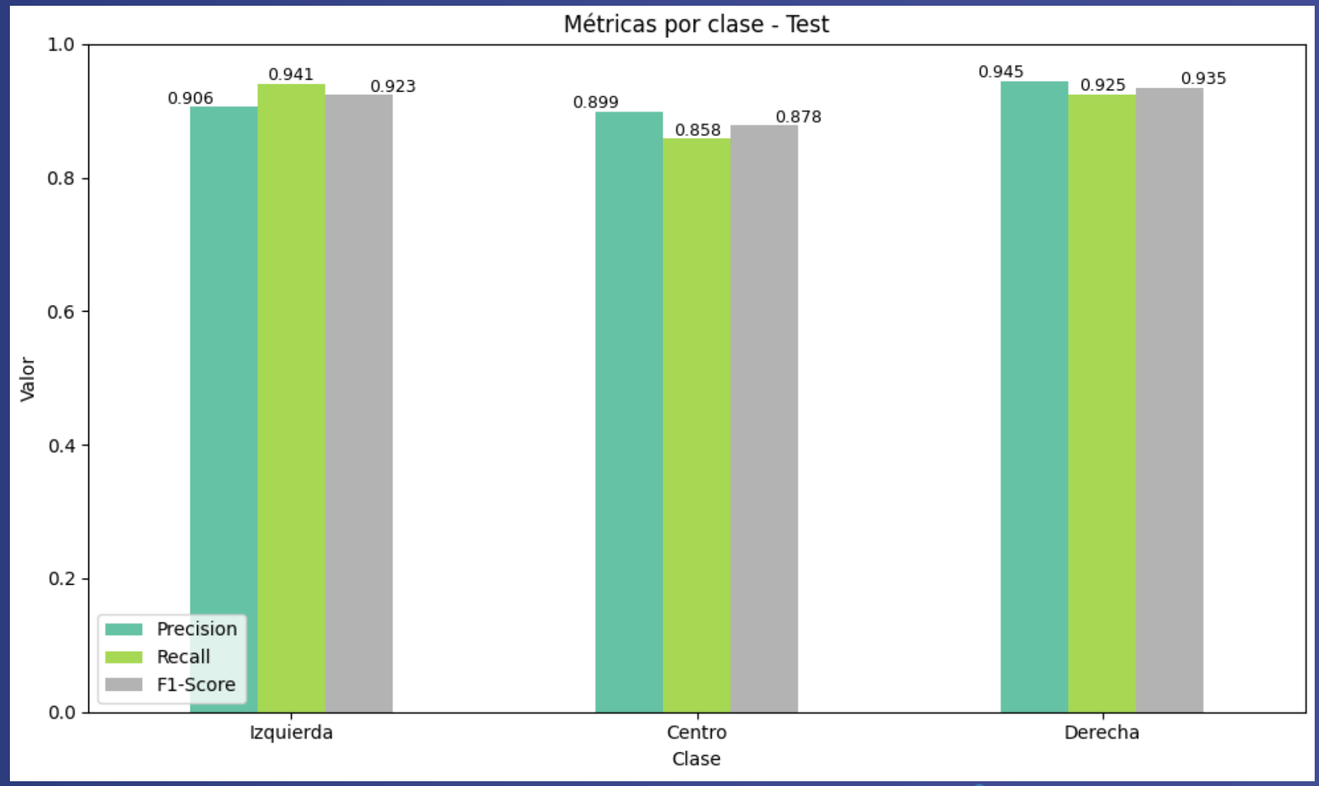
```
📊 Métricas finales sobre el conjunto de TEST:
Accuracy:  0.9167
F1 Score:  0.9165
Precision: 0.9169
Recall:    0.9167
```



Siguiente →

Resultados finales





Siguiente →

Resultados finales

En todos los gráficos e imágenes anteriores, vimos que nuestro modelo logro ser realmente robusto frente a los datos a los que nunca había tenido acceso (test dataset). Como equipo estamos sumamente contentos con el modelo que logramos y apuntamos a poder agregar un feature que permita al modelo recibir artículos en ingles por un tercero y poder hacer una clasificación del mismo.

Siguiente →

Aplicacion final

Finalmente, se pudo incluir la funcionalidad para agregar url de artículos de los Estados Unidos de noticias periodisticas, dejando los siguientes resultados que reflejan lo robusto del sistema obtenido.

```
[13] url_1 = 'https://nypost.com/2025/06/09/opinion/its-a-small-part-of-la-is-latest-media-refrain-to-minimize-violent-mobs/' #Ejemplo derecha

[17] prediccion = predecir_sesgo(url_1, model, tokenizer, bias_labels)
     print("El sesgo político del artículo es:", prediccion)

     El sesgo político del artículo es: Derecha

[29] url_2 = 'https://abcnews.go.com/US/trump-la-siege-mayor-governor-paint-picture/story?id=122652268' #Ejemplo Izquierda

[30] prediccion = predecir_sesgo(url_2, model, tokenizer, bias_labels)
     print("El sesgo político del artículo es:", prediccion)

     El sesgo político del artículo es: Izquierda

[40] url_3 = 'https://www.bbc.com/news/articles/c78e7gjz387o'

     prediccion = predecir_sesgo(url_3, model, tokenizer, bias_labels) #Ejemplo Centro
     print("El sesgo político del artículo es:", prediccion)

     El sesgo político del artículo es: Centro
```

Siguiente →

Gracias por su tiempo