PoliticES

Detección de Ideología Política

Integrantes del grupo: Álvaro Martín Bacas y Eva Temes Moya
Asignatura: Procesamiento del Lenguaje Natural
Curso académico: 2024/2025
Universidad: Universidad de Jaén
Fecha de entrega: 11 de mayo 2025

INDICE

1. Introducción

1.1 Objetivo de la práctica

El objetivo principal de esta práctica es d**iseñar, implementar y evalua**r un sistema de **Procesamiento del Lenguaje Natura**l capaz de clasificar la **ideología política** expresada en publicaciones escritas en español procedentes de la red social X. Esta tarea implica **aplicar un flujo completo** de trabajo en PLN, que incluye análisis de datos, preprocesamiento textual, diseño del sistema, entrenamiento y evaluación de modelos de clasificación.

El **sistema desarrollado** ha sido evaluado con tweets no utilizados durante el entrenamiento, empleando la métrica **macro-F1** como criterio de evaluación. Los resultados de cada ejecución se han subido a **Kaggle**, lo que permite comparar el rendimiento del modelo con el de otros participantes en la competición.

1.2 Motivación del problema

En los últimos años, las **redes sociales** se han consolidado como uno de los principales espacios de **expresión y discusión política**. Plataformas como X concentran diariamente millones de opiniones sobre temas sociales, económicos y culturales. Ante este volumen masivo de contenido, contar con **herramientas automáticas** que permitan identificar la **orientación ideológica** de los mensajes —por ejemplo, si expresan posturas afines a la izquierda o la derecha— puede ser de **gran utilidad** para estudios de polarización, análisis del discurso, seguimiento de campañas políticas o detección de tendencias emergentes.

Esta práctica propone **construir un sistema** capaz de analizar publicaciones breves, informales y a menudo ambiguas, como los tweets, y **determinar su posicionamiento ideológico**. Se trata de un **reto técnico y lingüístico** dentro del campo del procesamiento del lenguaje natural (PLN).

1.3 Descripción del conjunto de datos PoliticES

Para esta tarea se ha utilizado el conjunto de datos **PoliticES**, un corpus creado en el marco de una competición organizada dentro del workshop IberLEF 2023. Este conjunto incluye publicaciones reales en español de la red social X, anotadas con la ideología política que transmiten.

Los datos se dividen en dos partes:

- **Un archivo de entrenamiento**, que contiene publicaciones etiquetadas como left (izquierda) o right (derecha), y que hemos utilizado para entrenar y ajustar nuestro sistema.

- **Un archivo de test**, con publicaciones sin etiquetar, sobre las que se deben generar predicciones para la competición.

2. Análisis del conjunto de datos de train

Antes de entrenar cualquier modelo de clasificación, es **fundamental comprender** a fondo las **características del conjunto de datos** con el que se va a trabajar. En una **primera aproximación, pasamos por alto este análisis previo** y optamos por introducir directamente al modelo información como hashtags relacionados con partidos políticos, esperando que esto facilitara la tarea. Sin embargo, **los resultados fueron muy pobres**: el modelo tendía a sobre ajustarse a pistas superficiales y no lograba generalizar correctamente.

Esta experiencia subraya la importancia de explorar y entender bien los datos antes de entrenar cualquier sistema. Un **análisis adecuado** permite detectar posibles fuentes de

ruido, **desequilibrios** entre clases, **patrones** lingüísticos relevantes o decisiones de **preprocesamiento** que pueden tener un impacto significativo en el rendimiento.

A continuación, se presenta un **análisis detallado del conjunto de entrenamiento** utilizado, que sirvió como base para ajustar nuestra estrategia de modelado.

2.1. Estructura y características del conjunto de entrenamiento

El conjunto de entrenamiento contiene **180.000 publicaciones** reales en **español** extraídas de la red social X. Cada entrada tiene **dos columnas principales**: **text** (el contenido textual del post) y **label** (la ideología política: left o right).

Los textos presentan el **estilo típico de las redes sociales**: son breves, coloquiales y muchas veces contienen abreviaturas, hashtags, menciones, e incluso emojis. Esto añade **bastante ruido y complejidad a la tarea.**

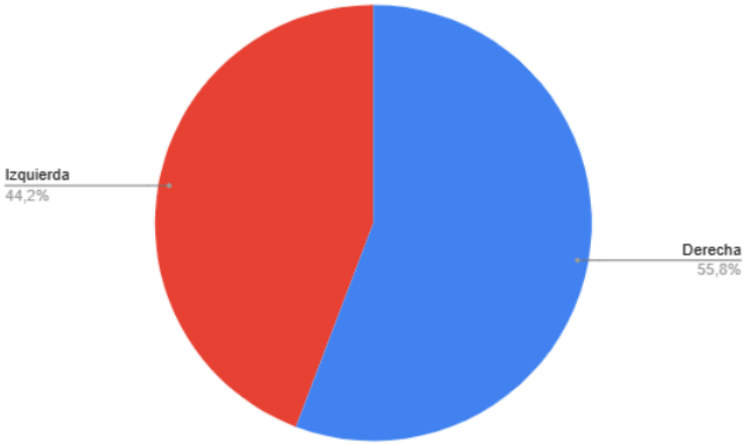
El conjunto de entrenamiento se presenta ya con ciertos **elementos anonimizados y estandarizados**, lo que afecta directamente a la forma en que el modelo interpreta el contenido. En particular, todas las **menciones a usuarios** han sido reemplazadas por el token genérico ***@user***, eliminando cualquier información específica sobre los emisores o destinatarios de los mensajes. Asimismo, los **hashtags** han sido sustituidos por la etiqueta ***[HASHTAG]***, ya que no contienen información relevante para la tarea.

Además, se ha **eliminado** cualquier **referencia** directa a **nombres de políticos o partidos**. En su lugar, se han introducido etiquetas genéricas como ***[POLITICIAN]*** para personas y ***[POLITICAL_PARTY]*** para formaciones políticas.

2.2. Distribución de clases

La distribución entre clases **no está totalmente equilibrada**:

- **Izquierda (left): 55.78%**
- **Derecha (right): 44.22%**



2.3. Estadísticas descriptivas de los textos

Analizando la **longitud de los textos**:

- **Número medio de palabras** por publicación: **28.76**
- **Desviación estándar: 12.79**
- **Longitud mínima: 4 palabras**
- **Longitud máxima: 65 palabras**

Percentiles:

- **25%** de los textos tienen **menos de 17 palabras**
- **50%** tienen **28 palabras**
- **75%** tienen **40 palabras o menos**

En cuanto a elementos especiales:

- **Emojis**: presentes en un 16.08% de los textos
- **Hashtags:** no hay # presentes prácticamente ya que la media es muy próxima a 0.
- **Menciones a usuarios**: hay 0.56 menciones de media por texto

2.4. Observaciones relevantes

Uno de los **principales retos** de trabajar con publicaciones en redes sociales es la **naturaleza del lenguaje utilizado**. Los textos son breves —con una media de aproximadamente 29 palabras—, lo que limita el contexto disponible para que el modelo infiera la ideología. Esta concisión exige que el sistema sea capaz de **captar rápidamente los elementos más informativos del mensaje.**

Además de ser cortos, los textos presentan **un lenguaje muy informal**: incluyen abreviaciones, expresiones coloquiales, emojis y menciones a otros usuarios. Este estilo introduce un alto nivel de ruido semántico, lo que dificulta el procesamiento con técnicas

tradicionales. Por ello, es crucial utilizar modelos que manejen bien lenguaje no estructurado, como los basados en arquitecturas Transformer con embeddings contextuales.

Otro aspecto clave es que **el conjunto de datos ya ha sido sometido a un proceso de anonimización y estandarización**. Nombres de políticos, partidos y hashtags han sido reemplazados por etiquetas genéricas, lo que **evita que el modelo se base en pistas superficiales**. Si bien esto favorece la generalización, también elimina información explícita que podría haber facilitado la clasificación. Esto obliga al sistema a centrarse en matices discursivos, tono e implicaturas ideológicas, para lo cual modelos como RoBERTa son especialmente adecuados, ya que capturan mejor el contexto.

La distribución de clases en el conjunto de entrenamiento no es completamente equilibrada: un 55.78% de las publicaciones se etiquetan como "izquierda" frente al 44.22% como "derecha". Aunque **el desbalance no es grave**, puede influir en el comportamiento del modelo, favoreciendo predicciones hacia la clase mayoritaria. Por ello, es importante aplicar estrategias como **el uso de la métrica macro-F1** para evaluar el rendimiento de manera equitativa, y considerar técnicas de balanceo como el submuestreo, sobremuestreo o la utilización de funciones de pérdida ponderadas.

En cuanto a la **longitud de los textos**, existe una alta **variabilidad**, con publicaciones que van desde las **4 hasta las 65 palabras**. Esto sugiere la necesidad de utilizar modelos que puedan adaptarse a entradas de diferente tamaño, y de aplicar algún tipo de normalización de longitud si se observa que influye en el rendimiento. La capacidad de adaptarse a secuencias cortas y ruidosas es fundamental para esta tarea.

También se debe tener en cuenta el uso de **elementos no textuales** como los emojis, presentes en más del 16% de las publicaciones. **Estos símbolos pueden aportar información afectiva o ideológica implícita,** por lo que conviene asegurarse de que el tokenizer y el modelo los procesen adecuadamente. Algunos modelos multilingües o preentrenados específicamente en redes sociales, como BETO, pueden manejar estos elementos mejor que modelos más generales.

El conjunto de datos PoliticES presenta, además, desafíos importantes en la clasificación ideológica debido a la **presencia frecuente de emociones, lenguaje subjetivo e ironía.** Estos aspectos dificultan considerablemente la tarea para modelos tradicionales como TF-IDF o Naive Bayes, que se basan en representaciones estadísticas simples y no tienen capacidad para capturar el contexto ni el subtexto ideológico implícito.

Para abordar esta complejidad, se recurre a modelos **basados en arquitecturas Transformer**, como RoBERTa, que permiten representar el significado de las palabras en función del contexto en el que aparecen. En este caso, se utiliza una versión preentrenada en español, **PlanTL-GOB-ES/roberta-base-bne**, que ha sido ajustada mediante *fine-tuning* para adaptarla al dominio específico de la tarea

3. Proceso y experimentación

Durante el desarrollo del sistema, realizamos una serie de **experimentos** aplicando distintos enfoques, desde **métodos clásicos de representación textual** hasta **modelos avanzados basados en Transformers**. Esta fase fue clave para comprender las particularidades del conjunto de datos **PoliticES** y tomar decisiones informadas sobre qué técnicas ofrecían el mejor rendimiento en la tarea de **detección de ideología política**.

A continuación, se detallan los enfoques más relevantes evaluados durante el proceso, explicando sus fundamentos, implementación, resultados obtenidos y las principales lecciones extraídas de cada uno.

Enfoque 1: Representación TF-IDF y características lingüísticas

El primer enfoque explorado fue el uso de **representaciones TF-IDF combinadas con características lingüísticas**. Esta estrategia pretendía establecer una línea base utilizando **métodos tradicionales de procesamiento de texto**. Se representaron los textos mediante vectores **TF-IDF**, incluyendo n-gramas de uno y dos términos, y se añadieron variables auxiliares como la **frecuencia de palabras ideológicamente marcadas**, la **presencia de signos de puntuación expresivos** y el **uso de emojis**. Para la clasificación, utilizamos modelos como **regresión logística** y **SVM**.

Aunque este enfoque ayudó a entender la estructura general del problema, los **resultados obtenidos no fueron los mejores**. El modelo no era capaz de captar el **contexto** ni el **subtexto ideológico**, alcanzando un **F1-score de 0.52**, lo que destacó la necesidad de usar modelos con mayor capacidad para entender relaciones semánticas y matices del lenguaje.

Enfoque 2: Modelo RoBERTuito con pesos de clase

Ante las limitaciones del enfoque anterior, utilizamos un **modelo Transformer preentrenado ligero**: **"robertuito-base-uncased"**, diseñado específicamente para textos en español. Dada la leve **desbalance de clases** en el conjunto de entrenamiento, incorporamos un **ajuste de pesos de clase** para minimizar su impacto. Este modelo mostró mejoras, especialmente en la captura de **estructuras lingüísticas complejas**. Sin embargo, al ser un modelo relativamente pequeño, su desempeño seguía siendo limitado frente a publicaciones con **ironía**, **referencias indirectas** o **sarcasmo**. Aunque mejoró el rendimiento, el **F1-score no alcanzó los valores deseados**, por lo que decidimos explorar modelos más potentes.

Enfoque 3: Modelos grandes basados en Transformers

Finalmente, centramos nuestra atención en **modelos más grandes y potentes**, como **RoBERTA**, **BETO** o **MarIA**, todos entrenados sobre grandes **corpus en español**. Estos modelos permiten realizar un **fine-tuning específico** para nuestra tarea,

beneficiándose de **embeddings contextuales** que capturan mejor el significado de las palabras en su contexto.

Tras varias pruebas, seleccionamos el modelo **"PlanTL-GOB-ES/roberta-base-bne"**, una versión de **RoBERTa** entrenada en español peninsular. Este modelo mostró un **rendimiento notable desde las primeras iteraciones**. Ajustamos **hiperparámetros** como la tasa de aprendizaje, el número de épocas, el tamaño del batch, y aplicamos técnicas como **early stopping** y **evaluación basada en F1-score**. Después de numerosas pruebas y ajustes, este modelo proporcionó los **mejores resultados**, alcanzando un **F1-score de 0.68871** en la competición de **Kaggle**. Finalmente, fue elegido como nuestra **solución final**.

4. Código final: arquitectura y configuración del modelo

4.1 Explicación del código

Después de una fase de experimentación exhaustiva, el modelo que mejores resultados ofreció fue RoBERTa preentrenado en español, específicamente la versión **PlanTL-GOB-ES/roberta-base-bne**, desarrollada en el marco del Plan de Tecnologías del Lenguaje del Gobierno de España. Este modelo destaca por estar entrenado con grandes volúmenes de datos textuales en español de España, lo que resultó clave para captar matices ideológicos, ironías y referencias contextuales propias del lenguaje político en redes sociales.

El código implementado sigue un pipeline clásico de fine-tuning sobre modelos de Transformers utilizando la biblioteca **transformers** de Hugging Face. Primero, se carga el conjunto de datos etiquetado en formato CSV, se extraen los textos y sus correspondientes etiquetas, y se realiza una división en conjuntos de entrenamiento y validación mediante **train_test_split** de Scikit-learn. Esta separación permite evaluar el modelo sobre ejemplos no vistos durante el entrenamiento, garantizando una evaluación más fiable.

El preprocesamiento se centra en la **tokenización** de los textos con el tokenizer asociado a RoBERTa. Cada publicación se convierte en una secuencia de identificadores numéricos, incluyendo las máscaras de atención necesarias para el modelo. La longitud máxima de secuencia se definió en **256 tokens**, un valor que ofrece un buen equilibrio entre retención de información y eficiencia computacional, especialmente dado que los textos en el dataset suelen ser breves.

Una vez tokenizados, los datos se convierten al formato **torch** y se agrupan en objetos **Dataset** para que puedan ser gestionados por el **Trainer** de Hugging Face. El modelo se entrena utilizando el objeto **TrainingArguments**, donde se configuran los hiperparámetros clave: tasa de aprendizaje de **3e-5**, **0.5 épocas**, tamaño de batch de **32**, pasos de calentamiento, decaimiento de peso, y activación de **fp16** para aprovechar la precisión mixta en GPU, acelerando el proceso de entrenamiento.

La métrica utilizada para seleccionar el mejor modelo durante el entrenamiento fue **F1-score**, ya que es la medida indicada en el guion de la práctica para evaluar el rendimiento y la que se buscó optimizar. Después de finalizar el entrenamiento, el modelo se evaluó sobre el conjunto de validación y se utilizó para generar predicciones sobre el conjunto de test proporcionado por la competición. Estas predicciones se guardaron en un archivo **predicciones.csv**, con el formato requerido para la evaluación (cabecera + 43,759 filas).

Es importante resaltar que **no se utilizó una estrategia de preprocesamiento exhaustivo** del texto. En pruebas previas, al eliminar **stopwords,** menciones, emojis, números o signos de puntuación, l**os resultados empeoraron**. Esto subraya la importancia de no eliminar ciertos elementos del lenguaje que pueden ser esenciales para capturar el contexto ideológico y emocional de las publicaciones.

En cuanto a la configuración específica del modelo, algunas decisiones clave fueron tomadas tras un análisis detallado del conjunto de datos y los recursos disponibles. La longitud máxima de secuencia se fijó en **max_length=256**, ya que la mayoría de las publicaciones tenían menos de 50 palabras, con una longitud media de 28. Este límite más amplio garantiza que no se trunque información en los casos más largos sin sacrificar la eficiencia computacional.

El número de **épocas** de entrenamiento se limitó a **0.5** para evitar el sobreajuste y acortar el tiempo de ejecución en un entorno con recursos limitados (Google Colab). Al partir de un modelo preentrenado, solo fue necesario ajustar ligeramente sus pesos. Para garantizar que el modelo final fuera el mejor, se utilizó la opción **load_best_model_at_end=True** y se realizaron evaluaciones periódicas con **eval_steps**, lo que permitió seleccionar el mejor checkpoint según el F1-score.

Finalmente, la activación de **fp16=True** (precisión mixta) fue una decisión clave, ya que permitió reducir significativamente el uso de memoria en GPU, acelerando el proceso de entrenamiento sin pérdidas sustanciales de rendimiento. Esto resultó ser especialmente útil en Google Colab, donde las capacidades de GPU son limitadas. Esta configuración también permitió experimentar con tamaños de batch relativamente altos (batch_size=32) sin agotar la memoria disponible.

4.1 Código final

```
import csv
import torch
import numpy as np
from transformers import RobertaForSequenceClassification, RobertaTokenizer, Trainer,
TrainingArguments
from datasets import Dataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support, accuracy_score

if torch.cuda.is_available():
    print(f"Usando la GPU: {torch.cuda.get_device_name(0)}")
else:
```

```
    print("CUDA no está disponible. El modelo se ejecutará en la CPU.")


ruta_entrenamiento = '/content/drive/MyDrive/PLN/PoliticES_train_kaggle.csv'
ruta_prueba = '/content/drive/MyDrive/PLN/PoliticES_test_kaggle.csv'
archivo_predicciones = '/content/drive/MyDrive/PLN/predicciones.csv'


model_name = "PlanTL-GOB-ES/roberta-base-bne"

tokenizer = RobertaTokenizer.from_pretrained(model_name)
model = RobertaForSequenceClassification.from_pretrained(model_name, num_labels=2)


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)


textos, tags = [], []
with open(ruta_entrenamiento, 'r', encoding='utf-8') as archivo_csv:
    lector_csv = csv.reader(archivo_csv, delimiter=',')
    next(lector_csv)  # Saltar encabezado
    for fila in lector_csv:
        if fila:
            textos.append(fila[1])
            tags.append(0 if fila[2] == 'left' else 1)

# Dividir los datos en entrenamiento y validación
train_texts, val_texts, train_labels, val_labels = train_test_split(textos, tags, test_size=0.1,
random_state=42)

# Crear datasets
train_dataset = Dataset.from_dict({"text": train_texts, "label": train_labels})
val_dataset = Dataset.from_dict({"text": val_texts, "label": val_labels})

# Tokenizar los textos
def tokenizar_lote(batch):
    return tokenizer(batch["text"], truncation=True, padding="max_length", max_length=256)

train_dataset = train_dataset.map(tokenizar_lote, batched=True)
val_dataset = val_dataset.map(tokenizar_lote, batched=True)

train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
val_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])

# Métricas para el entrenador
def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)
    return {'accuracy': acc, 'f1': f1, 'precision': precision, 'recall': recall}

training_args = TrainingArguments(
    output_dir="./resultados",
    eval_strategy="steps",  # Evaluación cada ciertos pasos
    save_strategy="steps",  # Guardar el modelo cada ciertos pasos
    eval_steps=2000,  # Evaluar cada 1000 pasos
    save_steps=2000,  # Guardar cada 1000 pasos
    logging_dir="./logs",
    logging_steps=250,
    learning_rate=3e-5,
    num_train_epochs=0.5,
```

```
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    warmup_steps=500,
    weight_decay=0.01,
    fp16=True if torch.cuda.is_available() else False,
    load_best_model_at_end=True,
    metric_for_best_model="f1"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

# Entrenar el modelo
trainer.train()

# Evaluar el modelo
results = trainer.evaluate()
print("Resultados de evaluación:", results)

# Procesar datos de prueba
textos_a_predecir, ids_a_predecir = [], []
with open(ruta_prueba, 'r', encoding='utf-8') as archivo_csv:
    lector_csv = csv.reader(archivo_csv, delimiter=',')
    next(lector_csv)  # Saltar encabezado
    for fila in lector_csv:
        if fila:
            textos_a_predecir.append(fila[1])
            ids_a_predecir.append(fila[0])

# Crear dataset de prueba
test_dataset = Dataset.from_dict({"text": textos_a_predecir})
test_dataset = test_dataset.map(tokenizar_lote, batched=True)
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask"])

# Realizar predicciones
predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=-1)

# Archivo de salida
with open(archivo_predicciones, 'w', encoding='utf-8', newline='') as archivo_salida:
    escritor_csv = csv.writer(archivo_salida)
    escritor_csv.writerow(["id", "label"])
    for i in range(len(ids_a_predecir)):
        escritor_csv.writerow([ids_a_predecir[i], "left" if preds[i] == 0 else "right"])

print(f"Predicciones guardadas en {archivo_predicciones}")
```

5. Conclusión

A lo largo de esta práctica, hemos diseñado, implementado y evaluado un **sistema de clasificación de ideología política** basado en técnicas de **Procesamiento del Lenguaje Natural** y **modelos de aprendizaje profundo**. Partimos de un análisis

exhaustivo del conjunto de datos **PoliticES**, explorando distintas aproximaciones, desde métodos tradicionales como **TF-IDF** hasta arquitecturas avanzadas como los **modelos Transformers entrenados específicamente en español**.

Durante el desarrollo, **probamos una gran variedad de modelos, estrategias de preprocesamiento y configuraciones de entrenamiento**. Incluso planteamos la hibridación de varios **modelos grandes basados en Transformers** para combinar sus fortalezas, pero no disponíamos del tiempo ni de los recursos necesarios para implementarlo. Aun así, creemos que esta última línea habría ofrecido un **rendimiento aún mejor**. El **modelo final**, basado en **PlanTL-GOB-ES/roberta-base-bne**, alcanzó un **F1-score competitivo de 0.68518**.

Cabe destacar que la mayoría de pruebas fueron realizadas en el entorno de **Google Colab**, usando aceleración por hardware **Tesla T4**, ya que estos modelos se benefician mucho de la memoria GPU. Una vez agotados los recursos de la versión gratuita, o cuando necesitábamos más tiempo de ejecución, usamos **Anaconda** en uno de los ordenadores con una **tarjeta gráfica NVIDIA**, ya que en otro caso los tiempos de entrenamiento llegaban a durar más de **20 horas**. Por ello, en el código se comprueba si se está trabajando con **CUDA**.

El proceso ha sido exigente: dedicamos muchas horas a **experimentar, analizar errores y ajustar parámetros**, enfrentándonos por primera vez a un **volumen de datos** y una **complejidad computacional** de esta escala. Consideramos que, más allá del resultado final obtenido, esta práctica nos ha permitido **aprender sobre técnicas más avanzadas en PLN** y descubrir varias formas de afrontar el mismo problema.