Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos originales
Temática: Inteligencia artificial
Recibido: 16/04/2023 | Aceptado: 15/07/2023 | Publicado: 30/09/2023

Identificadores persistentes:
DOI: 10.48168/innosoft.s12.a99
ARK: ark:/42411/s12/a99
PURL: 42411/s12/a99

Reconocimiento y Clasificación de Mensajes de Odio

Recognition and Classification of Hate Messages

Patrick Leopoldo Paredes Neira [1*], Gary Jamil Vilca Tapia [2], Kristhyan Andree Kurt Lazarte Zubia [3]

[1] Universidad La Salle, Arequipa, Perú. pparedesn@ulasalle.edu.pe
[2] Universidad La Salle, Arequipa, Perú. gvilcat@ulasalle.edu.pe
[3] Universidad La Salle, Arequipa, Perú. klazartez@ulasalle.edu.pe

[*] Autor para correspondencia: pparedesn@ulasalle.edu.pe

Resumen

El uso masivo de las redes sociales y el anonimato que este brinda ha posibilitado no solamente la comunicación inmediata entre los usuarios, sino también que acrezca la difusión del discurso de odio contra ciertos grupos de nuestra sociedad en forma de mensajes ofensivos para ellos, esto ha desembocado en un grave problema social; el cual sigue siendo tema de investigación actual junto con NLP. El propósito del presente trabajo es hacer una comparación de nuestro modelo de reconocimiento de "HateCheck" contra los resultados del autor, utilizando la misma base de datos que ellos. Para ello haremos uso de las principales métricas como son: precisión, recall y F1.

Palabras clave: Odio, HateCheck, Modelos de Detección del Discurso de Odio, NLP, Procesamiento del Lenguaje Natural, Discurso de Odio, Procesamiento del Español.

Abstract

The massive use of social networks and the anonymity that this provides has made possible not only the immediate communication between users, but also the spread of hate speech against certain groups of our society in the form of offensive messages to them, this has led to a serious social problem, which remains a topic of current research along with NLP. The purpose of the present work is to make a comparison of our "HateCheck" recognition model against the author's results, using the same database as them. To do so, we will make use of the main metrics such as: precision, recall and F1.

Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

Keywords: *Hate, HateCheck, Hate Speech Detection Models, NLP, Natural Language Processing, Hate Speech, Spanish Processing.*

I. Motivación

A. ¿En qué dominio del conocimiento está trabajando?

Basándonos en nuestro problema propuesto y nuestro objetivo principal, podemos concluir que el dominio que estamos abarco es el de la Sociología, debido a que abordaremos el estudio de una muestra de la sociedad como un conjunto de relaciones cultural y los mensajes que estos escriben para su posterior clasificación.

B. ¿Quiénes son los usuarios objetivo?

En general, cualquier persona que pueda acceder a un medio de comunicación a través de la internet, por ejemplo: foros, redes sociales, etc. Debido a que a través de toda la información recopilada podremos clasificar de manera más eficiente los mensajes que inciten al odio. Para ello haremos una amplia búsqueda de diferentes bases de datos que se detallarán posteriormente.

C. ¿Por qué es interesante el tema que proponen?

Tenemos que tener en cuenta que la libertad de expresión es uno de los derechos que todos poseemos por el simple hecho que somos seres humanos. Pero nuestros derechos acaban cuando se atenta contra los de otro individuo, y de todas las formas de comunicación que existen en la actualidad, la más popular son los mensajes de texto, los cuales lo usamos en todo momento (documentos, foros, redes sociales, correos electrónicos, etc.), es por ello que resulta agotador pensar en toda la información que se tiene que procesar por una persona, en lugar de un ordenador de forma sistemática. Además, luego de clasificar estos mensajes se podrá ver quiénes fueron los usuarios que tienen más inclinación a esos pensamientos para su posterior tratamiento. Incluso, saber cuáles son los temas que provocan una sensación de odio en las personas y su reducción en los medios de comunicación.

D. ¿Cuáles son las preguntas que su proyecto de NLPintenta responder?

Nuestro proyecto intenta responder sobre cuan viable sería implementar un modelo de reconocimiento de frases de odio en español para su posterior clasificación, así como obtener de forma cuantitativa la precisión de este, teniendo en cuenta que normalmente los modelos son entrenados para el reconocimiento del inglés y que los investigadores de NLP tanto en inglés como en otros idiomas suelen tener como lengua materna el inglés.

- ¿Para qué me serviría usar esta herramienta de HateCheck?

- ¿Se puede mejorar el sesgo de rendimiento con respecto a otras herramientas?

II. Problema

Definimos a los mensajes de odio o a la incitación al odio como un abuso dirigido contra los miembros que conforman un determinado grupo, estos están basados en la edad, discapacidad, identidad de género, la raza, los orígenes nacionales o étnicos, la religión, la orientación sexual (haciendo foco a estos últimos), lo que puede generar una sería de discrepancias de ideología con personas que no comparten esas idas o no pertenecen a dicho grupo. Este problema está íntimamente relacionado a la discriminación racial.

Es por tales motivos y basándonos en estas definiciones, enfocamos la detección de la incitación al odio como la clasificación binaria de contenidos como odiosos o no odiosos.

III. Objetivo

El presente trabajo tiene como finalidad el poder utilizar una Inteligencia Artificial y entrenarla con un dataset de diferentes frases de odio en español, categorizados según el grupo social al cual va dirigido para que esta pueda reconocer, discernir y clasificar si un texto es de odio. Para ello tendremos 7 categorías: personas negras, homosexuales, indígenas, judías, discapacitadas, transexuales y mujeres.

Asimismo, nuestra implementación será comparada con los resultados del equipo de investigación del artículo "HateCheck: Functional Test for Hate Speech Detection Models" bajo las 3 principales métricas conocidas: precisión, recall y F1 [1].

IV. Datos

A. ¿Qué datos necesitará?

Para la realización del presente proyecto se ha visto necesario tener un dataset el cual no solamente contenga los mensajes de odio, sino que también debe estar clasificado en 7 categorías: personas negras, homosexuales, indígenas, judías, discapacitadas, transexuales y mujeres.

B. ¿Cómo recolectarán los datos?

Debido a las limitaciones de nuestros recursos, optaremos por usar una data en español que sea de uso libre cuyos autores pertenezcan a una organización o universidad de prestigio.

C. ¿Dónde planea obtenerlos?

La búsqueda lo haremos en los principales repositorios de bases de datos como: Kaggle, Hugging Face, Google, UCI Machine Learning Repository y GitHub. Se referenciará apropiadamente a los autores.

D. ¿Cómo planea almacenarlos?

Los datos serán llevados a un archivo de Excel con extensión .xls para que puedan ser facilmente visualizados y posteriormente manipulados.

E. ¿Cómo accederá a ellos para utilizarlos en su proyecto?

Ya que trabajaremos con Python y el notebook Jupyter (Google Colab), usaremos la librería Pandas y Numpy que nos permitirá la lectura de la data y su obtención en una estructura de datos como diccionarios y matrices, para su posterior procesamiento.

V. Diseño

A. ¿Cómo sugiere usar la herramienta para ayudar a los usuarios a realizar las tareas que respondan a las preguntas que enumeró en la motivación?

El usuario deberá cargar una archivo Excel con extensión .xls o en su defecto deberá tipear los mensajes que desea clasificar, luego nuestro "HateCheck" le asignará su respectiva categoría para finalmente devolver al usuario otro archivo .xls.

Al usar esta herramienta poder comprender los abusos más comunes descritos en la industria de la tipografía también poder tener diagnosticos más especificos con "HateCheck" ya que cuenta con pruebas funcionales, y también es ampliamente usada en la detección de discursos del odio [3], a diferencia de Bert que clasifica erróneamente los comentarios relacionados a mujeres más que en otras categorías.

El sesgo de rendimiento del "HateCheck" podemos mejorarlo de manera significativa evitando el uso de palabras soeces en expresiones exclamativas. También podemos mejorarla evitando usar negaciones antes de ingresar cada mensaje [2].

VI. Trabajos relacionados

- Strong negative emotions explicitly expressed about a protected group or its members: Resembles "expressed hatred" [5]
- Sobre el lenguaje ofensivo Palmer et al. (2020) [7] recopilan tres datasets para evaluar el rendimiento del modelo en lo que denominan lenguaje ofensivo complejo, donde se utilizan los insultos, adjetivos y distanciamiento lingüistico.

VII. Revisión Literaria

En este presente trabajo se centrará específicamente en los problemas de clasificación de texto que contienen lenguaje ofensivo. Durante el transcurso del proyecto encontramos problemas que afectan la precisión y la eficacia del modelo que estamos desarrollando, dentro de los cuales nos encontramos con: el dominio del idioma, las métricas comparativas de modelos y la clasificación que reciben algunas de las palabras que solo son soeces mas no ofensivas.

Para poder identificar los enfoques que abordan algunos de estos problemas se requirió de la revisión de artículos que se encuentran disponibles en la página de la IEEE, además de ello daremos una breve descripción de cada enfoque realizado por diferentes trabajos.

Para nuestro primer enfoque dará solución al problema de la eficiencia del modelo por ello usaremos un enfoque basado en pruebas, la necesidad de este enfoque es porque los modelos de detección tienen debilidades que no se pueden observar por medio de las métricas. El beneficio de usar este modelo para nuestra investigación es que cuenta con bastantes pruebas funcionales que nos ayudarán a clasificar de mejor manera la información, este enfoque se basa principalmente en el reconocimiento de caracteristicas que presentan gran dependencia a muchas palabras específicas. [9].

En segundo lugar, tenemos el enfoque que nos ayuda a tener un contexto claro al momento de hacer la clasificación de mensajes que inciten al odio. Para abordar este desafío se ha propuesto agregar tareas adicionales al corpus de entrenamiento, como la predicción binaria y la multietiqueta. La necesidad de este enfoque surgió ya que dentro de las redes sociales se usa el sarcasmo. Para nuestro estudio esto representa un factor clave al momento de procesar información [14].

En tercer lugar, el idioma representa un factor importante al momento de hacer la clasificación de información, esto se debe a que tenemos limitaciones al momento de hacer un análisis de comentarios en una red social famosa como por ejemplo twitter. La necesidad de este enfoque es debido a que nuestro modelo solamente está basado en el idioma inglés, en vista a esta necesidad se desarrolla el multilingual hatecheck que cuenta con idiomas variados y mejoras en sus pruebas funcionales. Pero esta herramienta cuenta con errores que serán cubiertos en su mayoría, errores como el corpus del modelo al contener tantos idiomas disminuye la certeza de sus resultados [15].

Finalmente encontramos un último enfoque que se basa en mezclar varias técnicas para mejorar la precisión de los datos de salida del modelo. La necesidad de este enfoque es cuando tenemos como un conjunto de entrada datos que provienen de redes sociales entonces lo que se espera es contar con un corpus grande. Las técnicas encontradas son el bigrama

que usa de representación de características BOW, después encontramos la técnica de los diccionarios que están basados en N-gramas y finalmente nos encontramos con diccionarios que nos serviran para generar características maestras. Para concluir hemos podido encontrar a través de la literatura un enfoque de aprendizaje por transferencia basados en modelos preentrenados de tipo BERT que utilizan las técnicas anteriormente mencionadas
[16].

La elección de los enfoques depende del contexto en el cuál vamos a querer usar el modelo. Si los datos de entrada fueran de redes sociales usaríamos la herramienta multilingual, en caso quisiéramos contar con una mayor precisión usariamos el enfoque basado en técnicas.

Para finalizar dirémos que en todo modelo hemos encontrado muchos puntos débiles es por ello que los enfoques nos ayudan a fortalecer esos puntos débiles. Como se comentó al inicio, existen problemas que no podemos darles solución como es el caso de comparar dos modelos, ya que las métricas no nos ayudan a compararlos. Se puede concluir que para realizar el trabajo debemos contar con un corpus lo suficientemente grande para realizar una clasificación más precisa.

VIII. Diseño

A. Propuesta

Para este trabajo se utilizó un dataset de frases relacionadas con mensajes de odio que están previamente categorizadas en 7 tipos dependiendo al grupo social al que se refiere: homosexuales, personas de color, transexuales, indígenas, judíos, discapacitados, mujeres y también está la categoría de que no es un mensaje de odio. Como ventaja de este dataset es que ya se le realizó preprocesamiento y limpieza de emojis, pero mantiene las formas alternativas de palabras de odio que reemplazan ciertas letras por números o que las palabras están separadas por espacios.

Con respecto a los pasos de tokenización, tags y lemas se utilizará SpaCy junto con el modelo preentrenado es_dep_news_trf

Sobre el proceso de reconocimiento de entidades con nombre (Named Entity Recognition, NER) también se utilizará SpaCy junto con su modelo preentrenado con esta tarea, en caso de que no se obtenga buenos resultados se optará por el uso de otras alternativas como BERT.

Finalmente, para la clasificación usaremos BERT y afinaremos sus parámetros y usaremos etiquetas para que realice un entrenamiento supervisado a fin de obtener buenos resultados.

Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

B. Pipeline

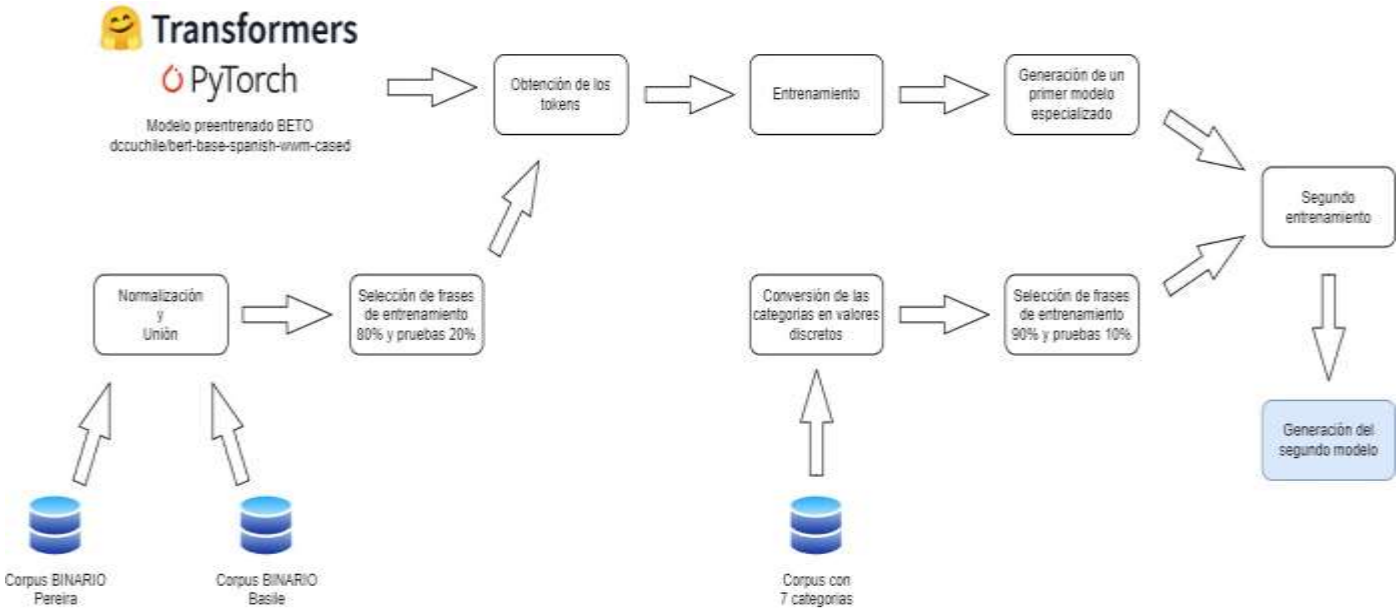

Fig. 1: Pipeline del proceso de clasificación de mensajes de odio

C. Principios básicos de diseño que nos guió para crearlo

Los principios de procesamiento que están presentes son los principios estándares para el reconocimiento de texto antes de ser procesados, de forma que generamos las características del modelo BERT. Las reglas aplicadas al corpus son las siguientes:

- Pasar todo el texto a minúsculas.
- Remover URLs contenidas en el texto.
- Remover emojis que contenga el dataset.
- Remover puntuación.
- Remover saltos de línea innecesarios.

La selección de las características y la parametrización de los modelos se realizó de acuerdo a la revisión de la literatura. Se procedió a realizar una comparativa de la información de los estudios revisado. En ello se identificó la utilización del modelo SVM con características de tipo unigrama para establecer el punto de referencia con los demás modelos en comparación. La siguiente tabla contempla los modelos utilizados y los mejores resultados obtenidos según la literatura.

TABLE I: Comparación de métodos

Estudio	(Paul, et al, 2020)	(Esma, et al, 2022)
Modelo	Modelo Propuesto%	CNN static y word2vec
Precisión	0.6443	Sin información
Recall	0.9628	Sin información
F1 score	- -	89.6
Resultados	Basados en unigramas	CNN simple de una capa

D. Fortalezas y debilidades del diseño

1) Fortalezas: Nuestro diseño frece varias ventajas significativas para el procesamiento de lenguaje natural (NLP) en el reconocimiento de mensajes de odio. Aquí se mencionan algunas:

- Contextualización bidireccional:

  Nuestro diseño captura el contexto y la relación entre las palabras en una oración de manera bidireccional, gracias a BERT. Esto significa que comprende tanto las palabras anteriores como las posteriores en un texto, lo que mejora la comprensión del significado y la intención detrás de las palabras clave relacionadas con el odio. Esta contextualización mejorada ayuda a identificar mensajes de odio más precisamente.

- Representaciones lingüísticas variadas:

  Nuestro diseño será entrenado en un dataset ya clasificado, lo que le permite aprender patrones y características del lenguaje de manera objetiva. Como resultado, el diseño desarrolla representaciones lingüísticas ricas que capturan la semántica y el significado de las palabras, con un previo análisis morflógico. Esto es beneficioso para el reconocimiento de mensajes de odio, ya que puede captar sutilezas en el lenguaje que indican contenido ofensivo.

- Adaptabilidad a diferentes idiomas:

  El diseño será entrenado y aplicado al idioma español, pero tenemos datasets de diferentes idios lo que lo hace adaptable a diferentes lenguajes para su reconocimiento y clasificación. Esto es especialmente valioso para el reconocimiento de mensajes de odio, ya que el contenido ofensivo puede estar presente en diferentes idiomas y culturas.

Estas ventajas hacen que nuestra propuesta sea una opción versátil para el procesamiento de lenguaje natural en el reconocimiento de mensajes de odio, aunque es importante considerar que ningún modelo es perfecto y siempre se deben tener en cuenta las limitaciones y los posibles sesgos inherentes en los datos y el entrenamiento.

2) Debilidades:

- Manejo de texto estructurado: Los mensajes de odio a menudo se presentan en forma de texto no estructurado, como comentarios en redes sociales o publicaciones en foros. Nuestro planteamiento no es efectivo en el procesamiento de este tipo de texto, ya que apesar de capturar el contexto y el significado gramatical más informal, esto se ve afectada por su capacidad en relación al testing y entrenamiento ya que será a partir de un dataset estructurado y previamente clasificado.
- Requerimientos computacionales y de recursos: Es un modelo de lenguaje profundo y complejo que requiere una cantidad significativa de recursos computacionales y memoria para su entrenamiento y aplicación (BERT). Esto dificulta su implementación en entornos con recursos limitados, lo que limita su accesibilidad y uso en algunos casos.
- Tamaño del modelo: Dependiendo del tamaño del modelo y del data set que usamos, principalmente con una relacion de 10 000 palabras puede presentar un desafío para el modelo de reconocimiento y clasificación de mensajes de odio.
- Dependencia de datos etiquetados: Aunque BERT se puede adaptar a tareas específicas con datos etiquetados limitados, aún requiere una cantidad significativa de datos anotados para un mejor desempeño. La recopilación y el etiquetado de grandes conjuntos de datos pueden ser costosos y consume tiempo, especialmente cuando se trata de mensajes de odio, que a menudo requieren una revisión manual exhaustiva.

Es importante tener en cuenta estas desventajas y considerarlas en el desarrollo y la implementación de modelos de procesamiento de lenguaje natural para el reconocimiento de mensajes de odio. Además, es crucial realizar una evaluación constante.

IX. Código

Ahora, se mostrará el paso a paso para implementar este sistema de reconocimiento y clasificación de mensajes de odio en español:

A. Importamos las bibliotecas y habilitamos el uso de CUDA

Para empezar con la implementación tenemosque importartensorflow y como usamos la GPU para acelerar el proceso también debemos de habilitar el uso de CUDA, se obtiene el nombre de la GPU. Limitamos el usode la VRAM, debemos de tener en cuenta de que la GPU sea compatible con CUDA y si no tenemos alguna se utilizará la CPU.

Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

```
import tensorflow as tf

# Obtener el nombre del dispositivo GPU. device_name =
tf.test.gpu_device_name()

# El nombre del dispositivo de GPU debe ser similar al
,→siguiente:
if device_name == '/device:GPU:0':
      print('Found GPU at: ', device_name)
else: raise SystemError('GPU device not found')
```

Fig. 2: Código Python

```
import torch if
torch.cuda.is_available():
      # Apuntamos a la primera GPU CUDA
      #torch.cuda.set_per_process_memory_fraction(gpu_fraction) device =
      torch.device("cuda") print('# GPUs: ',
      torch.cuda.device_count()) print('Nombre de la GPU:',
      ,→torch.cuda.get_device_name(0))

else:
      print('Se utilizará la CPU') device =
      torch.device("CPU")

model_str = "dccuchile/bert-base-spanish-wwm-cased"
```

Fig. 3: Código Python

B. Leemos el corpus

Para leer el corpus de datos usamos pandas y se lee el siguiente archivo. CORPUS DE DATOS Posteriormente se realiza la tokenización; es decir, se le agregar un token específico a cada palabra para poder analizarla.

```
import pandas as pd df=
pd.read_excel('corpus_bin.xlsx')
df.groupby("HS").size() #1 -> Hate speech
```

Fig. 4: Código Python

```
train_df = df.sample(frac=0.8, random_state=2023) test_df =
df.drop(train_df.index) train_df.head()
```

Fig. 5: Código Python

```
from transformers import BertTokenizer, ,→
BertForSequenceClassification, AdamW tokenizer =
BertTokenizer.from_pretrained(model_str)
```

Fig. 6: Código Python

Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

C. Tokenizamos el texto y obtenemos las etiquetas

En este paso se combierte el texto en secuencias numéricas uusando el tokenizador BERT estableciendo un máximo de 512 por texto. Esto puede variar y lo podemos modificar para disminuirlo y aumentaría el tiempo de ejecución.

```
max_length = 512

def encode_text(texts):
   input_ids = []
   attention_masks = [] for text
   in texts:
      encoded_text = tokenizer.encode_plus(
            text, add_special_tokens=True,
            max_length=max_length,
            truncation=True,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt'
      ) input_ids.append(encoded_text['input_ids'])
attention_masks.append(encoded_text['attention_mask']) input_ids
= torch.cat(input_ids, dim=0)
```

Fig. 7: Código Python

D. Cargamos el modelo y los detalles para entrenar

En este paso secarga el modelo pre entrenado se le da como parámetro el modelo y el número de CAT del modelo pre entrenado

```
model = BertForSequenceClassification.from_pretrained(
      model_str, num_labels=2,

)
```

Fig. 8: Código Python

E. Realizamos el entrenamiento

Para el entrenamiento por épocas, se establece el número que se va a ciclar su entrenamiento, luego se ingresa la data completa y luego se transforma las entiquetas en el tipo de dato necesario en nuestro caso convertimos a tipo entero, en vectores binarios, a tipo flotatnte. Este mismo proceso se repite para convertir las predicciones a tipo entero, luego se obtiene la clase con mayor probabilidad, esto para ejemplo. También debemo usar las métricas de sklearn para calculas la precisión, el f1, el recall y la precisión para cada lote o época.

```
for n_epoca in range(epocas): print(f"### Epocas: {n_epoca + 1}
    de {epocas}
    ,→ ###\n")

    train_loss = 0.0 train_accuracy =
    0.0 train_f1 = 0.0 train_recall =
    0.0 train_precision = 0.0
    model.train()

    for batch in train_dataloader: batch_input_ids
        = ,→ batch['input_ids'].to(device)
        batch_attention_mask =
        ,→ batch['attention_mask'].to(device) batch_labels =
        batch['labels'].to(device)

        # Convertir las etiquetas a tipo entero batch_labels =
        batch_labels.long()

        # Convertir las etiquetas en vectores binarios batch_labels
        = F.one_hot(batch_labels, ,→ num_classes=N_CAT)
        model.zero_grad()

        # Convertir las etiquetas a tipo flotante batch_labels =
        batch_labels.float()

        outputs = model(
            batch_input_ids, token_type_ids=None,
            attention_mask=batch_attention_mask,
            labels=batch_labels
        )

        loss = outputs[0] logits =
        outputs[1] train_loss +=
        loss.item()

        labels = batch_labels.to('cpu').numpy() preds =
        logits.detach().cpu().numpy()

        # Convertir las etiquetas y las predicciones a
        ,→ tipo entero
        labels = labels.astype(int) preds =
        preds.astype(int)

        # Obtener la clase con mayor probabilidad para
        ,→ cada ejemplo
        labels = np.argmax(labels, axis=1) preds =
        np.argmax(preds, axis=1)

        # Usar las métricas de sklearn para calcular
        ,→ la precisión, el f1, el recall y la
        ,→ precisión para cada lote
        train_accuracy += accuracy_score(labels,
        ,→ preds) train_f1 += f1_score(labels, preds, ,→
        average='macro', zero_division=0) train_recall +=
        recall_score(labels, preds,
        ,→ average='macro', zero_division=0) train_precision +=
        precision_score(labels, ,→ preds, average='macro',
        zero_division=0) loss.backward()

        adam.step() scheduler.step()
```

Fig. 9: Código Python

```
avg_train_loss = train_loss /
↪ len(train_dataloader) avg_train_accuracy =
train_accuracy /
↪ len(train_dataloader) avg_train_f1 = train_f1 /
len(train_dataloader) avg_train_recall = train_recall /
↪ len(train_dataloader) avg_train_precision =
train_precision / ↪ len(train_dataloader)

# Imprimir y dar formato a las métricas con dos
↪ decimales
print(f'Train loss: {avg_train_loss:.2f}') print(f'Train accuracy:
↪ {avg_train_accuracy:.2f}') print(f'Train f1: {avg_train_f1:.2f}')
print(f'Train recall: {avg_train_recall:.2f}') print(f'Train precision:
↪ {avg_train_precision:.2f}') print('-' * 30)

val_loss = 0.0 val_accuracy = 0.0
val_f1 = 0.0 val_recall = 0.0
val_precision = 0.0 model.eval()

for batch in test_dataloader: batch_input_ids = ↪
    batch['input_ids'].to(device)
    batch_attention_mask =
    ↪ batch['attention_mask'].to(device) batch_labels =
    batch['labels'].to(device)

    # Convertir las etiquetas a tipo entero batch_labels =
    batch_labels.long()

    # Convertir las etiquetas en vectores binarios batch_labels =
    F.one_hot(batch_labels, ↪ num_classes=N_CAT)

    # Convertir las etiquetas a tipo flotante batch_labels =
    batch_labels.float()

    with torch.no_grad(): outputs = model(
            batch_input_ids, token_type_ids=None,
            attention_mask=batch_attention_mask,
            labels=batch_labels
        )

        loss = outputs[0] logits =
        outputs[1] val_loss += loss.item()

        preds = logits.detach().cpu().numpy() labels =
        batch_labels.to('cpu').numpy()
```

Fig. 10: Código Python

Revista Innovación y Software
Vol. 4, No. 2, Mes Septiembre-Febrero, 2023
ISSN: 2708-0935
Pág. 79-95
https://revistas.ulasalle.edu.pe/innosoft

```
        # Convertir las etiquetas y las
        ,→ predicciones a tipo entero
labels = labels.astype(int) preds =
preds.astype(int)

# Obtener la clase con mayor probabilidad
,→ para cada ejemplo
labels = np.argmax(labels, axis=1) preds =
np.argmax(preds, axis=1)

# Usar las métricas de sklearn para
,→ calcular la precisión, el f1, el
,→ recall y la precisión para cada lote
val_accuracy += accuracy_score(labels,
,→ preds) val_f1 += f1_score(labels, preds, ,→
average='macro', zero_division=0) val_recall +=
recall_score(labels, preds,
,→ average='macro', zero_division=0) val_precision +=
precision_score(labels,
,→ preds, average='macro', ,→
 zero_division=0)


avg_val_loss = val_loss / len(test_dataloader) avg_val_accuracy
= val_accuracy /
,→ len(test_dataloader) avg_val_f1 = val_f1 /
len(test_dataloader) avg_val_recall = val_recall /
len(test_dataloader) avg_val_precision = val_precision / ,→
len(test_dataloader)

# Imprimir y dar formato a las métricas con dos
,→ decimales
print(f'Validation loss: {avg_val_loss:.2f}') print(f'Validation
accuracy:
,→ {avg_val_accuracy:.2f}') print(f'Validation f1: {avg_val_f1:.2f}')
print(f'Validation recall: {avg_val_recall:.2f}') print(f'Validation
precision: ,→ {avg_val_precision:.2f}') print('\n')
```

Fig. 11: Código Python

X. Resultados Obtenidos

Luego de realizar el entrenamiento por época del sistema de reconocimiento de mensajes de odio, obtenemos resultados que sorprendentemente superaron a la implementaciónde [16], la cual hace uso de uno de los clasificadores más conocidos, ya que ha demostrado ser muy eficaz y precisión en la clasificación de textos. Una ventaja de ese clasificador es que suele funcionar bien incluso con una pequeña de datos de entrenamiento, además su dataset que ellos usaron para sus pruebas (testing y training) se pueden encontrar en las siguientes enlaces: HatEval y HateNet no dió referencia

de cuál escogió. Pero en [16] se menciona que en el caso del conjunto de datos HaterNet, los tres modelos más precisos (mBERT, XLM y BETO) predijeron las mismas etiquetas erróneas 54 veces de 600 y 207 de 1600 en el caso del conjunto de datos HatEval. En cuanto al mejor sistema BETO, las instancias mal etiquetadas están se inclinan hacia FP en el caso del conjunto de datos HatEval (71,03en el conjunto de datos HaterNet (61,32 Respecto a nuestro resultado obtenido, hemos aumentado y superado el valor obtenido por [16], a pesar que estamos usando un dataset con menos registros. Por otro lado, el aumento de lo obtenido se debe a qué antes de la clasificación estamos realizando una clasificación binaria del dataset, extrayendo todos los mensajes que contienen odio de los que no la tienen, eliminando los falsos positivos y aumentando la precision del modelo LaSalleBHS. Luego entrenamos el modelo con el dataset en español y lo entrenamos en un total de 7 épocas, luego de ello subimos el mode al Drive para su uso en una interfaz más usable.

A. Comparación de Resultados

Para la comparación de los resultados, estamos tomando como referencia [16] y [10] de tal forma que tomamos dos modelos, uno más antiguo que el otro. El primero es BETO, un modelo basado en BERT pero netamente entrenado para el idioma español. Y el segundo se trata de una Red Neuronal Convolucional (CNN en inglés), clasifica mensajes en español según 5 categorías: Homosexuales, personas de color, indígenas, judíos, mujeres y también está la categoría de que no es un mensaje de odio.

TABLE II: Comparación de métodos: BETO

Estudio	(Paul, et al, 2020)	LaSalleBHS
Modelo	BETO	LaSalle Beto hatespeach spanish
Precisión	78.87	1.00
Recall	76.51	1.00
F1 score	77.23	1.00
Pérdida	NA	0.00
Dataset	HaterNet	aCORPUS DE DATOS BINARIO

TABLE III: Comparación de métodos: CNN

Estudio	(Sun, et al, 2019)	LaSalleBHS
Modelo	GloVe + CNN	LaSalle Beto hatespeach spanish
Precisión	74.20	1.00
Recall	65.27	1.00
F1 score	67.20	1.00
Pérdida	NA	0.00
Dataset	HaterNet	aCORPUS DE DATOS BINARIO

XI. Conclusiones

Inicialmente, utilizamos un modelo BERT multilingüe y luego aplicamos un modelo de detección de discurso de odio creado en BETO. Sin embargo, el modelo BERT presenta errores en la clasificación. Al etiquetar una frase, BERT asigna tres etiquetas a una palabra que puede ser considerada como una mala palabra o malsonante. Además, cuando el corpus de entrenamiento no contiene nuevas palabras, como en nuestro caso, por ejemplo, la palabra "muerdealmohadas", el clasificador no podrá encontrar su significado y clasificará incorrectamente la frase. Por otro lado, BETO realiza un etiquetado más preciso, asignando una única etiqueta cuando encuentra una palabra ofensiva o malsonante. Esto evita confusiones durante la clasificación y mejora las métricas, como la precisión, F1, entre otras [4].

XII. Trabajos Futuros

Se espera mejorar la precisión de nuestro modelo de tal modo que pueda hacer la clasificación a través de un corpus más grande y permitir hacer una predicción más exacta haciendo uso de una variación de LTSM bidireccional [4].

Contribución de Autoría

Patrick Leopoldo Paredes Neira: Conceptualización, Análisis formal, Investigación, Visualización, Metodología, Software, Validación, Redacción - borrador original, Curación de datos, Escritura, revisión y edición. **Gary Jamil Vilca Tapia:** Conceptualización, Análisis formal, Investigación, Visualización, Metodología, Software, Validación, Redacción - borrador original, Curación de datos, Escritura, revisión y edición. **Kristhyan Andree Kurt Lazarte Zubia:** Conceptualización, Análisis formal, Investigación, Visualización, Metodología, Software, Validación, Redacción - borrador original, Curación de datos, Escritura, revisión y edición.

Referencias

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] VAN AKEN, Betty, et al., “Challenges for toxic comment classification: An in-depth error analysis“, 2018.

[3] BANKO, Michele; MACKEEN, Brendon, “RAY, Laurie. A unified taxonomy of harmful content. En Proceedings of the fourth workshop on online abuse and harms“, 2020, p. 125-137.

[4] PLAZA-DEL-ARCO, Flor Miriam, et al. “Comparing pretrained language models for Spanish hate speech detection. Expert Systems with Applications“, 2021, vol. 166, p. 114120.

[5] Thomas Davidson, Dana Warmsley, Michael Macy and Ingmar Weber "Automated hate speech detection and the problem of offensive languaje". In Proceedings of the 11th International AAAI Conference on Web and Social Media, 2017, p. 512-515

[6] Alexis Palmer, Christine Carr, Melissa Robinson, Jordan Sanders. "COLD: Annotation scheme and evaluation data set for complex offensive language in English", Journal for Language Technology and Computational Linguistics, p. 1-28.

[7] Alexis Palmer, Christine Carr, Melissa Robinson, Jordan Sanders. "COLD: Annotation scheme and evaluation data set for complex offensive language in English", Journal for Language Technology and Computational Linguistics, p. 1-28.

[8] B. Lohitha, M. V and J. J. Amarnath, "A Comparison of Different Models for the Detection of Hate Speech," 2022 1st International Conference on Computational Science and Technology (ICCST), CHENNAI, India, 2022, pp. 492-496, doi: 10.1109/ICCST55948.2022.10040400.

[9] RÖTTGER, Paul, et al. HateCheck: Functional tests for hate speech detection models. arXiv preprint arXiv:2012.15606, 2020.

[10] BALKIR, Esma, et al. Necessity and sufficiency for explaining text classifiers: A case study in hate speech detection. arXiv preprint arXiv:2205.03302, 2022.

[11] OTTER, Daniel W.; MEDINA, Julian R.; KALITA, Jugal K. A survey of the usages of deep learning for natural language processing. IEEE transactions on neural networks and learning systems, 2020, vol. 32, no 2, p. 604-624.

[12] WU, Stephen, et al. Deep learning in clinical natural language processing: a methodical review. Journal of the American Medical Informatics Association, 2020, vol. 27, no 3, p. 457-470.

[13] SHISHAH, Wesam; FAJRI, Ricky Maulana. Large Comparative Study of Recent Computational Approach in Automatic Hate Speech Detection. TEM Journal, 2022, vol. 11, no 1, p. 82.

[14] PÉREZ, Juan Manuel, et al. Assessing the impact of contextual information in hate speech detection. IEEE Access, 2023, vol. 11, p. 30575-30590.

[15] RÖTTGER, Paul, et al. MULTILINGUAL HATECHECK: Functional Tests for Multilingual Hate Speech Detection Models. arXiv preprint arXiv:2206.09917, 2022.

[16] ABRO, Sindhu, et al. Automatic hate speech detection using machine learning: A comparative study. International Journal of Advanced Computer Science and Applications, 2020, vol. 11, no 8.