Detección de Discurso de Odio Contra la Comunidad LGTBI+ Mexicana Mediante el Uso de Arquitecturas Transformer

Resolución de la tarea HOMO-MEX 2023

Carlos Fernández Rosauro

Grado en Ciencia de Datos Aplicada
Procesamiento del Lenguaje Natural

Tutor/a de TF
Montse Cuadros Oller
Profesor/a responsable de la asignatura
David Merino Arranz

Fecha Entrega
18/06/2023

Universitat Oberta de Catalunya

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Ficha del Trabajo Final

Título del trabajo:	Detección de Discurso de Odio Contra la Comunidad LGTBI+ Mexicana Mediante el Uso de Arquitecturas Transformer
Nombre del autor/a:	Carlos Fernández Rosauro
Nombre del Tutor/a de TF:	Montse Cuadros Oller
Nombre del/de la PRA:	David Merino Arranz
Fecha de entrega:	06/2023
Titulación o programa:	Grado en Ciencia de Datos Aplicada
Área del Trabajo Final:	Procesamiento del Lenguaje Natural
Idioma del trabajo:	Castellano
Palabras clave	NLP, Text Classification, Hate Speech

Resumen del Trabajo

El objetivo de este Trabajo de Fin de Grado es contribuir en la búsqueda de la igualdad para las personas del colectivo LGTBI+ a través de la detección de contenido LGTBI-fóbico en el contexto de las redes sociales. Para ello se trabaja sobre dos subtareas organizadas bajo el nombre de HOMO-MEX por parte de la conferencia IberLEF 2023. La primera subtarea consiste en un problema de clasificación multiclase para detectar tweets LGTBI-fóbicos, mientras que la segunda subtarea consiste en detectar el tipo concreto de LGTBI-fobia que exhiben estos tweets a través de un problema de clasificación multi-etiqueta. La resolución de las dos subtareas se ha abordado como problemas de clasificación de texto mediante técnicas básicas y avanzadas de Procesamiento de Lenguaje Natural. En concreto, se han utilizado modelos umbral de tipo estadístico, así como modelos Transformer basados en arquitecturas BERT y similares para ambas subtareas. Los modelos Transformer han obtenido unos resultados excelentes en la fase de validación tanto en el entorno experimental, como en la clasificación oficial de la tarea, donde el modelo RoBERTuito obtuvo la segunda posición en ambas subtareas. El modelo umbral Linear SVC de tipo tradicional y de menor coste computacional obtuvo resultados muy similares a los de los modelos Transformer en el contexto experimental, dando

espacio a la utilización de modelos más simples en condiciones de textos como las de las redes sociales.

Abstract

The goal of this thesis is to contribute to the improvement of equality for people from the LGTBI+ collective through the detection of LGTB-phobic content in the context of social networks. The technical side of the project is defined by two subtasks organized under the name of HOMO-MEX by the IberLEF 2023 conference. The first subtask consists of a multiclass classification problem to detect LGTBI-phobic tweets, while the second subtask consists of detecting the type specific LGTB-phobia exhibited by the LGTB-phobic tweets through a multilabel classification problem. The two subtasks have been tackled as text classification problems with both basic and advanced Natural Language Processing techniques. More specifically, baseline models, as well as Transformer models based on BERT and similar architectures have been used for both subtasks. The Transformer models have obtained excellent results in the validation phase both in the experimental environment and in the classification leaderboard of the task, as the RoBERTuito model obtained the second position in both subtasks. The Linear SVC baseline model of the traditional type and with lower computational cost obtained very similar results to those of the Transformer models in the experimental context, giving space to the use of simpler models in text classification tasks applied to social networks.

Index

Lista de Figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Este Trabajo de Fin de Grado trata de contribuir en la búsqueda de la igualdad para las personas del colectivo LGTBI+ a través de la detección de contenido LGTBI-fóbico en el contexto de las redes sociales. Según el Social Media Safety Index generado por la organización GLAAD, en 2021 un 64% de los usuarios LGTBI+ sufrieron acoso y/o fueron víctimas de discursos de odio [1]. Actualmente las compañías responsables de estas redes sociales no destinan suficientes recursos para evitar la propagación de contenido LGTBI-fóbico [2]. El principal resultado que se quiere obtener es aportar visibilidad a las posibilidades que ofrecen las nuevas técnicas y herramientas en el ámbito del Procesamiento del Lenguaje Natural a la hora de detectar mensajes de odio, prestando especial atención a los modelos conocidos como Transformer [3], ampliamente utilizados en la actualidad para tareas tan diversas como la creación de chatbots [4] o la resolución de problemas científicos de gran complejidad [5].

1.2. Objetivos del Trabajo

1. Explorar el rendimiento de los modelos Transformer en la detección de mensajes de odio.
2. Comparar el rendimiento de los modelos Transformer al de los modelos clásicos en la detección de mensajes de odio.
3. Participar en la competición HOMO-MEX 2023 y publicar un paper con la metodología y resultados.
4. Publicar el código utilizado para asegurar la reproducibilidad en tareas similares.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Haciendo referencia a la Guía transversal sobre la Competencia Ética y Global aportada por la dirección del programa, se definen los siguientes impactos en el diseño del proyecto:

1) Sobre la dimensión sostenibilidad:
A priori se prevé encontrar diferencias entre los distintos modelos de clasificación de textos utilizados en términos de coste energético en fases de implementación y despliegue. Esto queda alineado con el ODS 9 - Industry, innovation and infrastructure y más concretamente con el objetivo 9.4 - Upgrade all industries and infrastructures for sustainability [6].

2) Sobre la dimensión ético-social:
El desarrollo y publicación de este proyecto plantea un posible impacto negativo a través de la utilización de los conjuntos de datos proporcionados por la organización de la tarea HOMO-MEX. Es posible que a través de estos tweets fuese posible re-identificar a una persona individual y exponerla por el etiquetado que una persona y/o un algoritmo hayan

podido realizar sobre su comentario en la red social. Para mejorar el conjunto de anonimato y por lo tanto reducir la probabilidad de re-identificación de un usuario, se podrían seleccionar mensajes en redes sociales de cuentas anónimas, aunque esto podría sesgar el proceso de entrenamiento. En otros aspectos, el TFG no impacta en los ODS relacionados con esta dimensión.

3) Sobre la dimensión diversidad:
Este proyecto se enfoca claramente en la visibilización de la LGTBI-fobia y por lo tanto en la búsqueda de la diversidad. Se alinea plenamente con el ODS 10 - Reduced inequalities y más concretamente con el objetivo 10.2 - Promote universal, social, economic and political inclusion [7], ya que busca mejorar la inclusividad del colectivo LGTBI+ en las redes sociales.

1.4. Enfoque y método seguidos

A la hora de utilizar técnicas de Procesamiento del Lenguaje Natural para la visibilización de la problemática del discurso de odio contra el colectivo LGTBI+, existen fundamentalmente dos posibilidades:

A. Analizar sesgos de LGTBI-fobia en modelos de lenguaje populares como GPT-3 [8] o BERT [9].
B. Utilizar modelos del lenguaje para detectar mensajes LGTBI-fóbicos en textos.

La primera opción (A) constituye un problema de extrema complejidad porque a día de hoy resulta muy difícil interpretar el funcionamiento interno de los grandes modelos del lenguaje. La disciplina *mechanistic interpretability* se encarga de ello, aunque todavía está en sus inicios.

La segunda opción (B) resulta muy adecuada para adaptar y comparar técnicas clásicas y técnicas modernas del Procesamiento del Lenguaje Natural al problema de la clasificación de textos. El principal problema en este tipo de tarea proviene de la necesidad de un corpus de texto previamente etiquetado con el que trabajar. De esta forma, el enfoque que se ha seguido en este proyecto ha consistido en participar en la tarea HOMO-MEX [10] organizado por la conferencia IberLEF 2023 [11], donde, dados dos conjuntos de tweets en español mexicano, estos se deben clasificar mediante técnicas de minería de textos para detectar mensajes LGTBI-fóbicos y su tipología concreta. La tarea se abre al público a través de un repositorio propio en la plataforma Codalab [12].

1.5. Planificación del trabajo

Para el desarrollo del proyecto planteado fueron necesarios los siguientes recursos:

- Google Colab Pro como entorno de exploración de los datos y de desarrollo y validación de los modelos implementados.

- El lenguaje de programación utilizado fue Python. La librería de *deep learning* utilizada fue Tensorflow [13] y los modelos se accedieron a través de la comunidad Hugging Face [14] y se implementaron utilizando las herramientas que esta ofrece.
- El entorno de edición LaTeX del paper presentado en la tarea HOMO-MEX fue Overleaf [15], mientras que la memoria se redactó en Google Docs.
- Se utilizó Projectlibre [16] como software de gestión de proyectos para la asignación y revisión de tareas.

Las tareas planteadas para este TFG se agrupan en dos grandes bloques: Participación en la tarea HOMO-MEX y desarrollo técnico (1) y Documentación del TFG (2). Estos dos grandes bloques están a su vez relacionados de tal forma que todo el desarrollo técnico del proyecto planteado en el primer bloque debía irse incluyendo en los informes entregados como prácticas de evaluación continua (PEC) incluidas en el segundo bloque. A continuación se adjuntan las tareas planteadas como plan de trabajo de Projectlibre.

	Name	Duration	Start	Finish
1	**Participación en la tarea HOMO-MEX y desarrollo técnico**	**83 days**	**11/03/23 08:00**	**01/06/23 17:00**
2	Análisis, limpieza y procesamiento	5 days	11/03/23 08:00	15/03/23 17:00
3	Tokenización y selección de características	8 days	16/03/23 08:00	23/03/23 17:00
4	Entrenamiento y evaluación modelos simples	3 days	24/03/23 08:00	26/03/23 17:00
5	Investigación modelos estado del arte con Hugging Face	3 days	27/03/23 08:00	29/03/23 17:00
6	Entrenamiento y evaluación modelos Transformer	9 days	30/03/23 08:00	07/04/23 17:00
7	Finalización HOMO-MEX y envío de resultados	7 days	08/04/23 08:00	14/04/23 17:00
8	Mejora en la visualización de los datos de texto	15 days	01/05/23 08:00	15/05/23 17:00
9	Redacción y envío paper HOMO-MEX	16 days	17/05/23 08:00	01/06/23 17:00
10	**Documentación del TFG**	**100 days**	**11/03/23 08:00**	**18/06/23 17:00**
11	Preparación y entrega primer informe (PEC2)	25 days	11/03/23 08:00	14/04/23 17:00
12	Preparación y entrega segundo informe (PEC3)	40 days	17/04/23 08:00	26/05/23 17:00
13	Redacción y entrega de la memoria (PEC4)	20 days	30/05/23 08:00	18/06/23 17:00
Planificación TFG				

Figura 1: Planificación del proyecto por tareas.

1.6. Breve sumario de productos obtenidos

Como resultado de este proyecto se obtuvieron dos productos: un paper académico [Anexo 1] fruto de la participación en la competición HOMO-MEX 2023 y un repositorio en Github [17] con todo el código desarrollado incluyendo el procesamiento y limpieza de los datos y la implementación y validación de los modelos seleccionados.

1.7. Breve descripción de otros capítulos de la memoria

En el capítulo 2. Materiales y métodos se explica el desarrollo del proyecto desde el punto de vista técnico, de tal forma que en el apartado 2.1. Contextualización de la tarea, se explican los objetivos y metodologías a seguir de la tarea HOMO-MEX para cada una de las dos subtareas definidas (Track 1 y Track 2). En el apartado 2.2. Obtención, exploración y preparación de los datos se explican, exploran y preparan los conjuntos de entrenamiento ofrecidos por la organización de la tarea. En el apartado 2.3. Descripción de los modelos se introducen los modelos umbral clásicos y los modelos basados en arquitecturas Transformer que se implementarán, ejecutarán y validarán en el apartado 2.4. Experimentación, para cada una de las dos subtareas. En el capítulo 3. Resultados, se comentan de forma detallada los resultados obtenidos en el proceso de experimentación (3.1. Resultados analíticos), en la clasificación oficial de HOMO-MEX (3.2. Resultados de la competición) y adicionalmente se describen los productos obtenidos (3.3). El capítulo 4. Conclusiones y trabajos futuros recopila todas las conclusiones del proyecto y explora nuevas líneas de trabajo de potencial interés. El glosario se puede encontrar en el capítulo 5, la bibliografía en el capítulo 6 y los anexos en el capítulo 7.

2. Materiales y métodos

En este apartado se explicará la metodología seguida en el desarrollo de los modelos utilizados en la participación de la competición HOMO-MEX 2023, tal y como se expresa en el paper [Anexo 1] enviado a la organización IberLEF. También se explicará el proceso seguido para el desarrollo del propio paper, presentado junto con la tutora del Trabajo Montse Cuadros Oller.
Cabe destacar que todo el proceso de exploración de los datos, diseño, experimentación e implementación de los modelos, se desarrolló en el entorno de Google Colab, utilizando *notebooks* de Python. Para la exploración de los datos y el entrenamiento de los modelos umbral, se utilizó una instancia con CPU mientras que para el entrenamiento de los modelos Transformer, a excepción de mDeBERTaV3 [18], se utilizaron GPUs NVIDIA Tesla T4 o V100 y para el entrenamiento de mDeBERTaV3 se utilizó una GPU NVIDIA A100.

2.1. Contextualización de la tarea

En primer lugar se contextualiza la tarea HOMO-MEX en función de las instrucciones originales que se establecieron en la competición. Dado que la detección de los mensajes de odio resulta una tarea compleja y comprende varios niveles de abstracción, en esta competición se definieron dos subtareas o *tracks*:

- Track 1: detección de mensajes de odio como problema multiclase. En un problema multiclase [19] existen más de dos clases y cada muestra solo puede estar etiquetada con una de las clases. De esta forma, se toman tweets como datos de entrenamiento y el objetivo es indicar si exhiben contenido LGTBI-fóbico o no, teniendo que asignar a cada tweet una de las siguientes clases: LGTBI-fobia (P), no LGTBI-fobia (NP) o no relacionado con el colectivo LGTBI+ (NA).

- Track 2: detección detallada de mensajes de odio como problema multi-etiqueta. En un problema multi-etiqueta [19], cada muestra puede tener *m* etiquetas y cada una de las etiquetas puede tomar un valor entre *n* clases. En este caso, se toman tweets previamente ya clasificados como LGTBI-fóbicos y se deben etiquetar en función de su contenido. Las distintas etiquetas independientes que se trabajan en este problema son: lesbofobia (L), gayfobia (G), bifobia (B), transfobia (T) y/u otra LGTBI-fobia (O). Es importante recalcar que se tiene la flexibilidad de asignar más de una etiqueta a cada uno de los tweets, de tal forma que, por ejemplo, uno pueda tener asignado etiquetas de lesbofobia (L) y transfobia (T), mientras que otro tweet puede tener solamente asignada una etiqueta de bifobia (B).

En este punto resulta conveniente definir el concepto de LGTBI-fobia. Según el diccionario panhispánico del español jurídico [20], la LGTBI-fobia es el “rechazo, miedo, repudio, prejuicio o discriminación hacia mujeres u hombres que se reconocen a sí mismos como LGTBI”. Según el Observatorio Andaluz contra la Homofobia, Bifobia y Transfobia [21], “cuando hablamos de LGTBIFOBIA nos referimos a los hechos de intolerancia,

discriminación o rechazo a Lesbianas, Gays, Bisexuales y Transexuales por razones de orientación sexual o identidad de género".

Para la realización de las dos tareas presentadas, se entrenan y validan una serie de modelos de diferentes características que tienen como objetivo alcanzar la mejor capacidad predictiva posible. Para ello, definimos ahora las métricas que se tendrán en cuenta a la hora de evaluar los distintos modelos empleados. En ambas tareas tendremos en cuenta las métricas *F1-Score*, *Precision*, *Recall* y *Accuracy* calculadas sobre el subconjunto de validación (explicado en el apartado 2.2). Más concretamente, en la primera tarea utilizaremos la media ponderada de las métricas *F1-Score*, *Precision* y *Recall* porque esta tiene en cuenta la cantidad de ocurrencias de cada clase, lo cual resulta de gran importancia a la hora de evaluar un clasificador multiclase. En el caso de la segunda tarea, tendremos en cuenta la media macro de las métricas *F1-Score*, *Precision* y *Recall*, ya que esta trata de forma no ponderada la media de las distintas etiquetas, lo cual resulta imprescindible en el contexto de una clasificación multi-etiqueta, en la que las distintas etiquetas son independientes. En cualquier caso, se prestará especial atención a la métrica *F1-Score*, ya que es aquella que los/las organizadores/as de la tarea HOMO-MEX utilizaron para la evaluación de los modelos publicados en la clasificación global.

2.2. Obtención, exploración y preparación de los datos

Los conjuntos de datos para las dos tareas o *tracks* se obtuvieron de la web de la competición [12]. Para cada tarea se podía descargar un fichero de entrenamiento en formato csv. El fichero que contenía el conjunto de datos para la primera tarea tenía de esta forma 7000 tweets etiquetados con cada una de las tres categorías posibles: LGTBI-fobia (P), no LGTBI-fobia (NP) o no relacionado con el colectivo LGTBI+ (NA) (Figura 1). Además este conjunto de datos contenía también una columna índice para numerar los tweets. El fichero de la segunda tarea contenía un conjunto de 862 tweets LGTBI-fóbicos junto con 5 columnas que representaban cada una de las etiquetas asociadas a cada tweet de forma binaria (Figura 2), de tal forma que un valor igual a 1 correspondía a la etiqueta mostrando un valor positivo y un valor igual igual a 0 correspondía a la etiqueta mostrando un valor negativo.

Todo el proceso de exploración de los datos se realizó en el *notebook* TFG_EDA.ipynb. El primer paso fue importar los ficheros en el *notebook* de Google Colab y cargar cada uno de ellos como un DataFrame de Pandas [22]. A continuación se puede observar un pequeño fragmento de cada uno de los dos conjuntos de datos tras cargarlos como DataFrames con las columnas originales de los ficheros csv.

```
[ ]  task1_df.head()
```

	index	tweets	label
0	0	Me quise ligar a una chava ayer y no me pelo, ...	P
1	1	@papaya_rockera eres un puñal, Papayita.	P
2	2	Magnate ofrece 130 mdd al hombre que conquiste...	P
3	3	Los trolebuses del desgobierno de @EPN son idi...	P
4	4	En época de Hitler no se decía "eres gay" y, s...	P

Figura 2: Ejemplo del conjunto de datos de Track 1.

```
[ ]  task2_df.head()
```

	tuit	G	L	B	T	O
0	—Bueno, pues ya pasó la euforia papá, ahora sí...	1	0	0	0	0
1	Esos de BTS se me hacen seres asexuales, eunuc...	0	0	0	0	1
2	La partida de skywars que me ha hecho enojar m...	1	0	0	0	0
3	Se te acabo tu juego maldito joto asqueroso, e...	1	0	0	0	0
4	@Lunaazulada_ Será a los jotitos.??ellos prefi...	1	0	0	0	0

Figura 3: Ejemplo del conjunto de datos de Track 2.

También resulta de valor observar la distribución de las clases en cada uno de los DataFrames. Para ello podemos utilizar el método value_counts() de la librería Pandas. Cabe destacar que en el caso del DataFrame task1_df, asignado a la primera tarea, se sustituyó la clase tweet no relacionado con el colectivo LGTBI+ (NA) por un string 'NR' para una mayor facilidad de manipulación. A continuación se muestran las distribuciones de clases y etiquetas para cada tarea:

```
task1_df['label'].value_counts()
```

```
NP    4360
NR    1778
P      862
Name: label, dtype: int64
```

Figura 4: Distribución de clases en la Track 1.

```
task2_df[['G','L','B','T','O']].apply(pd.Series.value_counts)
```

	G	L	B	T	O
0	148	790	852	783	798
1	714	72	10	79	64

Figura 5: Distribución de etiquetas en la Track 2.

En la Figura 4 observamos que en la primera tarea, la gran mayoría de los tweets (clases NP y NR) no son LGTBI-fóbicos. Esto pone de manifiesto que el conjunto de datos está de base no balanceado, lo que significa que los tweets no están distribuídos de manera uniforme sobre las 3 clases. Esto supone que el entrenamiento de los modelos predictivos resultará complejo hacia la clase minoritaria (tweets LGTBI-fóbicos) al estar mal representada. En el caso de la figura 5, observamos que en la segunda tarea, la etiqueta que toma la gran mayoría de valores positivos es la de los tweets que exhiben odio hacia los hombres homosexuales (etiqueta G), mientras que el resto de etiquetas muestran claramente una frecuencia inferior en el conjunto de datos. Cabe destacar que la distribución conjunta de las distintas etiquetas no es importante y por lo tanto no se explora en este punto, ya que en el proceso de entrenamiento y predicción cada etiqueta se trabaja con una función de pérdida independiente.

Adicionalmente se calcularon la cantidad media de palabras que contenía cada tweet en cada una de las tareas. En la primera tarea (Track 1), cada tweet sin procesar contiene 22 palabras de media, mientras que en la segunda tarea (Track 2), cada tweet sin procesar contiene 16 palabras de media.

El procesado de los datos continúa con la implementación y aplicación de de una función procesamiento de textos. La función toma el nombre de process_tweets(tweet) y sigue los siguientes pasos para cada tweet procesado:

1. Convertir el texto a minúsculas y eliminar los espacios iniciales y finales del tweet.
2. Eliminar los símbolos de *retweet*, de *user handle* (@) y los hipervínculos.
3. Eliminar los *emojis* y los *hashtags*.
4. Eliminar los elementos numéricos individuales (números sueltos).
5. Eliminar *stopwords* cargadas desde la librería NLTK [23].
6. Quitar los puntos, comas, paréntesis y demás elementos de separación.
7. Implementar y aplicar un tokenizador de palabras de la librería NLTK.
8. Implementar y aplicar un Snowball *stemmer* [24] de la librería NLTK.

Es importante aclarar que esta función de procesado de los textos se aplica a los conjuntos de datos de ambas tareas para, en primer lugar, obtener una nube de términos que nos de una mejor visibilidad de la distribución de palabras útiles en el caso de cada tarea y, en

segundo lugar, para el entrenamiento de los modelos umbral. Este método de procesamiento de textos no se utilizará cuando se vayan a entrenar los modelos basados en arquitecturas Transformer, ya que dichos modelos cuentan con sus propios sistemas de tokenización y realizar algunas operaciones aplicadas en la función process_tweet() como eliminar *emojis*, *hashtags* o aplicar un *stemmer* podría eliminar información muy importante para un clasificador avanzado basado en redes neuronales.

Tras haber aplicado la función de procesamiento de tweets sobre los conjuntos de datos, podemos observar una muestra de los textos transformados:

```
tweets_task1
```

```
0       quis lig chav ayer pel pregunt si lesbian dio ...
1                                              puñal papayit
2            magnat ofrec mdd hombr conqu hij lesbian url
3       trolebus desgobiern idiot pon valeri dic pit t...
4       epoc hitl dec eres gay y eres jot camp concent...
                              ...
6995              igual cach transform tuit mir trist eso
6996            acab tempor rupauls drag rac dias regrets
6997    ayuñn pos dic oph send nerdez trans de " ahor ...
6998    si vam hac vagon segur inclu hombr gay saqu lench
6999    hmmm entiend reconoc diferent implic luch terc...
Name: tweets, Length: 7000, dtype: object
```

Figura 6: Tweets procesados de Track 1.

```
tweets_task2
```

```
0      buen pues pas eufori pap ahor dim vest blanc a...
1                        bts hac ser asexual eunuc asi rar
2      part skywars hech enoj mas ningun ven ak maric...
3                  acab jueg maldit jot asquer pag pq pag
4                            jotitosell prefier banderill
                              ...
857              part avil ching put madr gust puton ojet
858                                        acalambr putit
859    psst sol quier spoiler platic organiz ide cpmx...
860                              ps jot mejor vam cin parec
861       algui bien nen mariquit floj ademas quier acept
Name: tuit, Length: 862, dtype: object
```

Figura 7: Tweets procesados de Track 2.

Mediante la clase WordCloud de la librería wordcloud [25] podemos generar nubes de texto para cada uno de los conjuntos de datos procesados, representando en cada caso los 100 términos más frecuentes. En el caso de la primera tarea, tal y como se muestra en la

Figura 8, podemos observar bastantes términos relacionados con el colectivo LGTBI+, pero sin ver una tendencia clara. En cambio, con respecto a la segunda tarea, podemos observar en la Figura 9, como se observan muchos términos homófobos (o sus raíces) enfocados hacia hombres homosexuales: joto, puto, marica, etc. Estas figuras son una representación más visual de la distribución por clases y etiquetas que observamos anteriormente.

Figura 8: Nube de palabras para los tweets procesados (Track 1).

Figura 9: Nube de palabras para los tweets procesados (Track 2).

La fase de obtención, exploración y procesamiento de los datos finaliza con la separación de los conjuntos de datos de cada tarea en subconjuntos de entrenamiento y validación. Se optó por hacer una división simple a través de una semilla preestablecida sin tener en cuenta la situación de *class imbalance* con el objetivo de evaluar y comparar los modelos en un contexto de distribución desigual de clases y etiquetas. De esta forma, para la primera tarea se asignaron 5600 tweets al subconjunto de entrenamiento (80%) y 1400 al

subconjunto de validación (20%) de forma aleatoria, mientras que para la segunda tarea se asignaron 732 tweets (85%) al conjunto de entrenamiento y 132 (15%) al conjunto de validación también de forma aleatoria. La división se realizó mediante el método train_test_split() de sklearn [26].

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(tweets_task1, task1_df['label'], test_size=0.2, random_state=SEED)
```

Figura 10: Split en subconjuntos de entrenamiento y validación (Track 1).

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(tweets_task2, task2_df[['G','L','B','T','O']], test_size=0.15, random_state=SEED)
```

Figura 11: Split en subconjuntos de entrenamiento y validación (Track 2).

2.3. Descripción de los modelos

En ambas tareas se siguió la misma metodología: en primer lugar, se implementaron dos modelos predictivos clásicos como umbral y a continuación se implementaron una serie de modelos Transformer basados en arquitecturas BERT [9] para mejorar los resultados. Los modelos aplicados son los mismos pero adaptados al tipo de problema: clasificación multiclase en la primera tarea y clasificación multi-etiqueta en la segunda tarea.

2.3.1. Modelos umbral

Los modelos umbral son los modelos estadísticos de clasificación básicos que se implementan como punto de partida para la resolución de las tareas. El objetivo es mejorar los resultados obtenidos con estos modelos mediante modelos más avanzados.
Los modelos umbral son entrenados sobre matrices TF-IDF (*term frequency–inverse document frequency*) como variables *input* y las clases o etiquetas (dependiendo de la tarea) como variables *output*. Estas matrices TF-IDF se implementan mediante la clase TfidfVectorizer de la librería sklearn. En cada una de las tareas, el vectorizador se ajusta sobre el subconjunto de entrenamiento obtenido en el apartado 2.2 y después se transforma sobre el subconjunto de entrenamiento y el subconjunto de validación, obteniendo así dos matrices TF-IDF de entrenamiento y validación. A continuación, estas matrices TF-IDF entrenamiento y validación se utilizan para entrenar y validar los métodos umbral. Conviene recordar que los textos transformados en las matrices TF-IDF en este proceso son primero tratados mediante la función de procesamiento de textos process_tweets().

Se muestra primero la implementación de las matrices TF-IDF para cada tarea:

```
Tfidf_vect1 = TfidfVectorizer()
Tfidf_vect1.fit(X_train1)

Train_X_Tfidf1 = Tfidf_vect1.transform(X_train1)
Test_X_Tfidf1 = Tfidf_vect1.transform(X_test1)
```

Figura 12: Creación de matrices TF-IDF (Track 1).

```
Tfidf_vect2 = TfidfVectorizer()
Tfidf_vect2.fit(X_train2)

Train_X_Tfidf2 = Tfidf_vect2.transform(X_train2)
Test_X_Tfidf2 = Tfidf_vect2.transform(X_test2)
```

Figura 13: Creación de matrices TF-IDF (Track 2).

A. Multinomial Naive Bayes

El primero de los modelos umbral es un clasificador bayesiano ingenuo [27]. Este clasificador parte de aplicar el teorema de Bayes a la relación entre la variable dependiente y las variables independientes asumiendo la independencia condicional entre cada par de características (variables independientes, variable dependiente). En este caso la variable dependiente son las posibles clases que pueden tomar los tweets y las variables independientes o predictoras son los valores numéricos que toma la matriz TF-IDF.

B. Linear SVC

El segundo modelo umbral es un clasificador tipo 'Support Vector Machine' con un *kernel* lineal [28]. Este modelo permite clasificar cada tweet separando el espacio de variables independientes (valores de la matriz TF-IDF) mediante funciones lineales.

2.3.2. Modelos basados en Transformers

En este proyecto se puso el centro de atención sobre modelos de *deep learning* basados en arquitecturas BERT por dos motivos: el primero; su finalidad es perfecta para la tarea que nos ocupa, que es la de clasificación de textos, y el segundo; la disponibilidad de modelos pre entrenados en español y multilingües. De esta forma, se optó por trabajar con tres modelos que parten de la arquitectura BERT original o introducen mejoras.

A. BETO [29]. Se trata de un modelo del lenguaje basado en la arquitectura BERT original y pre entrenado completamente en español.
B. RoBERTuito [30]. Es una implementación de la arquitectura RoBERTa sobre textos de redes sociales y entrenado sobre 500 millones de tweets.
C. mDeBERTaV3 [18]. Es una implementación de la arquitectura DeBERTa desarrollada por Microsoft y entrenada sobre múltiples idiomas, entre los que se encuentra el español.

2.4. Experimentación

En este apartado se desarrolla para cada una de las tareas el proceso de implementación y evaluación de los modelos. Se comienza con los modelos umbral y después se procede con los modelos basados en arquitecturas BERT.

2.4.1. Track 1

En esta tarea, el objetivo es entrenar y validar cada uno de los modelos seleccionados sobre un problema multiclase, donde los inputs son una matriz TF-IDF en el caso de los modelos umbral o directamente los tweets en el caso de los modelos Transformer, y el output es la clase asignada.

A. Modelos umbral

Los dos modelos umbral se trabajan en el *notebook* TFG_EDA.ipynb. El modelo **Multinomial Naive Bayes** se implementa a través de la clase MultinomialNB [31] de la librería sklearn sobre la matriz TF-IDF de entrenamiento. Después se utiliza el modelo entrenado para calcular las predicciones sobre la matriz de TF-IDF de validación y finalmente podemos comparar estas predicciones con las clases originales para obtener las métricas.

La implementación del modelo y la obtención de las métricas se observan a continuación:

```
naive_bayes1 = MultinomialNB()
naive_bayes1.fit(Train_X_Tfidf1, y_train1)
predictions_NB1 = naive_bayes1.predict(Test_X_Tfidf1)
```

```
print(classification_report(predictions_NB1, y_test1))
```

```
              precision    recall  f1-score   support

          NP       0.99      0.66      0.80      1299
          NR       0.23      0.89      0.36        96
           P       0.03      0.80      0.05         5

    accuracy                           0.68      1400
   macro avg       0.42      0.78      0.40      1400
weighted avg       0.94      0.68      0.76      1400
```

Figura 14: Implementación y reporte de métricas Naive Bayes (Track 1).

En la Figura 14 observamos como el clasificador bayesiano obtiene unas métricas ponderadas *F1-Score* de 0.76, *Precision* de 0.94 y *Recall* de 0.68 y un *Accuracy* de 0.68.

De manera similar a como ocurría con Naive Bayes, el modelo **Linear SVC** se implementa a través de la clase SVC [32] de la librería sklearn sobre la matriz TF-IDF de

entrenamiento. Después se utiliza el modelo entrenado para calcular las predicciones sobre la matriz de TF-IDF de validación y finalmente podemos comparar estas predicciones con las clases originales para obtener de nuevo el reporte de métricas.

La implementación del modelo y la obtención de las métricas para **Linear SVC** se observan a continuación:

```
SVM1 = SVC(C=1.0, kernel='linear', gamma='auto')
SVM1.fit(Train_X_Tfidf1, y_train1)
predictions_SVM1 = SVM1.predict(Test_X_Tfidf1)
```

```
print(classification_report(predictions_SVM1, y_test1))
```

```
              precision    recall  f1-score   support

          NP       0.92      0.85      0.88       941
          NR       0.81      0.84      0.82       361
           P       0.40      0.64      0.49        98

    accuracy                           0.83      1400
   macro avg       0.71      0.78      0.73      1400
weighted avg       0.85      0.83      0.84      1400
```

Figura 15: Implementación y reporte de métricas Linear SVC (Track 1).

Así, en la Figura 15 quedan reflejadas las métricas ponderadas *F1-Score* de 0.84, *Precision* de 0.85 y *Recall* de 0.83 y *Accuracy* de 0.83 para el clasificador SVC lineal.

Mejorar las métricas obtenidas por los modelos umbral será el objetivo de los modelos Transformer.

B. Modelos Transformer

Para la implementación de todos los modelos Transformer se utilizaron las librerías datasets [33] y transformers [34] del ecosistema Hugging Face junto con la herramienta de computación y entrenamiento de modelos de *machine learning* Tensorflow. Esta configuración permite instanciar, entrenar y validar los modelos de una forma cómoda y reproducible. Aunque la metodología seguida para los distintos modelos resulta muy similar, se irán comentando las diferencias encontradas. Conviene recordar que los modelos Transformer utilizan sus propios métodos de procesamiento de texto antes de entrenar los clasificadores, de tal forma que la función process_tweets() no es aplicada a los tweets antes de utilizarlos como *inputs* en estos sistemas.

Antes de entrar en detalle en la configuración de los distintos modelos, resulta necesario describir aquellos parámetros e hiperparámetros que manualmente se han ajustado para mejorar el rendimiento de los modelos. Estos han sido: número de épocas (*epochs*),

tamaño de *batch* (*batch size*), tasas de aprendizaje (*learning rate*) inicial y final y probabilidad de *dropout*. Cuando en cada caso se establezcan aquellos hiperparámetros utilizados, estos fueron determinados tras un proceso intensivo de prueba y error manual. También resulta necesario destacar que en todos los casos se implementó un algoritmo de optimización AdamW [35] junto con un *learning rate scheduler* de tipo polinómico. En la Figura 16 se muestra un resumen de los hiperparámetros utilizados para cada modelo.

Model	Epochs	Batch size	start Learning rate	end Learning rate	dropout
BETO	4	4	$5e^{-5}$	0	-
RoBERTuito	7	4	$2e^{-5}$	0	0.2
mDeBERTaV3	5	32	$5e^{-5}$	0	0.2

Figura 16: Selección de hiperparámetros para modelos Transformer (Track 1).

Comenzando por el modelo **BETO** (notebook TFG_BETO_Track1.ipynb), realizamos una carga limpia de los datos de entrenamiento a partir del fichero csv original de la primera tarea en un Pandas DataFrame que se separa en los dos subconjuntos de entrenamiento y validación. Para asegurarnos que estos subconjuntos son siempre los mismos, utilizamos siempre la misma semilla al realizar el *split*.

```
task1_df = pd.read_csv(io.BytesIO(uploaded['homomex_training.csv']))

[ ] le = LabelEncoder()
    le.fit(task1_df['label'])

    ▾ LabelEncoder
    LabelEncoder()

[ ] list(le.classes_)

    ['NP', 'P', nan]

[ ] SEED = 2

[ ] train_texts, val_texts, train_labels, val_labels = train_test_split(list(task1_df['tweets']), list(le.transform(task1_df['label'])),
                                                                        test_size=.2, random_state=SEED, shuffle=True)

[ ] print(len(train_texts))
    print(len(val_texts))

    5600
    1400
```

Figura 17: Carga y split de datos para BETO (Track 1).

A continuación necesitamos descargar el *checkpoint* del modelo desde su repositorio en Hugging Face [36]. En el caso de BETO, utilizamos la versión bert-base-spanish-wwm-cased. Teniendo el *checkpoint* podemos construir un tokenizador para los tweets del subconjunto de entrenamiento y otro para los del subconjunto de validación. Cuando instanciamos los tokenizadores podemos añadir además al objeto cada clase asociada a cada tweet y de esta forma terminamos teniendo todo lo necesario para entrenar un sistema clasificador basado en una arquitectura BERT: 'input_ids', 'token_type_ids', 'attention_mask' y 'labels'.

```
checkpoint = "dccuchile/bert-base-spanish-wwm-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
Downloading (…)okenizer_config.json: 100% 364/364 [00:00<00:00, 22.0kB/s]
Downloading (…)lve/main/config.json: 100% 648/648 [00:00<00:00, 37.8kB/s]
Downloading (…)solve/main/vocab.txt: 100% 242k/242k [00:00<00:00, 1.80MB/s]
Downloading (…)/main/tokenizer.json: 100% 480k/480k [00:00<00:00, 2.46MB/s]
Downloading (…)cial_tokens_map.json: 100% 134/134 [00:00<00:00, 10.1kB/s]
```

```
[ ] train_encodings = tokenizer(train_texts, truncation=True, padding=True)
    val_encodings = tokenizer(val_texts, truncation=True, padding=True)
```

```
[ ] train_encodings['labels'] = train_labels
    val_encodings['labels'] = val_labels
```

```
[ ] train_encodings.keys()

    dict_keys(['input_ids', 'token_type_ids', 'attention_mask', 'labels'])
```

Figura 18: Descarga de *checkpoint* y generación de encodings para BETO (Track 1).

Dados los hiperparámetros ajustados *batch size* (4) y *epochs* (4) en el caso de BETO, ya podemos constituir los conjuntos de datos en formato de Tensorflow necesarios para el proceso de entrenamiento y validación.

```
batch_size = 4
num_epochs = 4
```

```
[ ] data_collator = DataCollatorWithPadding(tokenizer=tokenizer, return_tensors="tf")

    tf_train_dataset = Dataset.from_dict(train_encodings).to_tf_dataset(
        columns=["attention_mask", "input_ids", "token_type_ids"],
        label_cols=["labels"],
        shuffle=True,
        collate_fn=data_collator,
        batch_size=batch_size,
    )

    tf_validation_dataset = Dataset.from_dict(val_encodings).to_tf_dataset(
        columns=["attention_mask", "input_ids", "token_type_ids"],
        label_cols=["labels"],
        shuffle=False,
        collate_fn=data_collator,
        batch_size=batch_size,
    )
```

Figura 19: Creación de datasets Tensorflow para BETO (Track 1).

A continuación se implementa el algoritmo de optimización, con las características que ya se adelantaba anteriormente, y que en este caso comienza con un *learning rate* de $5e^{-5}$ y finaliza en 0. Teniendo en cuenta que se trata de un problema de clasificación multiclase,

se optó por una función de pérdida de tipo SparseCategoricalCrossentropy [37]. El modelo debe ser instanciado a través de la clase TFAutoModelForSequenceClassification de la librería transformers haciendo uso del *checkpoint* descargado y en este caso se utilizan las probabilidades por defecto para el dropout (0.1).

```
num_train_steps = len(tf_train_dataset) * num_epochs
lr_scheduler = PolynomialDecay(
    initial_learning_rate=5e-5, end_learning_rate=0.0, decay_steps=num_train_steps
)

opt = AdamW(learning_rate=lr_scheduler)
loss=SparseCategoricalCrossentropy(from_logits=True)
```

```
[ ] model = TFAutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=3,
                                                                 attention_probs_dropout_prob=0.1,
                                                                 hidden_dropout_prob=0.1,
                                                                 from_pt=True)
```

Figura 20: Implementación del optimizador, función de pérdida y modelo para BETO (Track 1).

Finalmente el modelo debe ser compilado haciendo referencia al optimizador y a la función de pérdida. Además, siempre añadimos la métrica de *Accuracy* sobre el conjunto de evaluación como guía para observar el avance del entrenamiento época a época. Tras la compilación, el modelo puede ser entrenado sobre el subconjunto de entrenamiento y validado sobre el subconjunto de validación.

```
model.compile(
    optimizer=opt,
    loss=loss,
    metrics=["accuracy"]
)
```

```
[ ] model.fit(
        tf_train_dataset,
        validation_data=tf_validation_dataset,
        epochs=num_epochs
    )

    Epoch 1/4
    1400/1400 [==============================] - 172s 88ms/step - loss: 0.5071 - accuracy: 0.8052 - val_loss: 0.4759 - val_accuracy: 0.8136
    Epoch 2/4
    1400/1400 [==============================] - 101s 72ms/step - loss: 0.3516 - accuracy: 0.8718 - val_loss: 0.4041 - val_accuracy: 0.8607
    Epoch 3/4
    1400/1400 [==============================] - 101s 72ms/step - loss: 0.1978 - accuracy: 0.9341 - val_loss: 0.4465 - val_accuracy: 0.8650
    Epoch 4/4
    1400/1400 [==============================] - 100s 72ms/step - loss: 0.0804 - accuracy: 0.9789 - val_loss: 0.5251 - val_accuracy: 0.8600
    <keras.callbacks.History at 0x7f9d00eb6fa0>
```

Figura 21: Compilación y entrenamiento de BETO (Track 1).

La evaluación del modelo se realiza comparando las predicciones sobre el subconjunto de validación con las clases originales de dicho subconjunto. De esta forma obtenemos unas métricas ponderadas *F1-Score* de 0.86, *Precision* de 0.87 y *Recall* de 0.86 y un *Accuracy* de 0.86 (Figura 22).

```
[ ]  preds = model.predict(tf_validation_dataset)["logits"]
     class_preds = np.argmax(preds, axis=1)

     350/350 [==============================] - 12s 27ms/step

[ ]  print(classification_report(class_preds, val_encodings['labels']))

                   precision    recall  f1-score   support

                0       0.92      0.89      0.90       899
                1       0.57      0.65      0.61       138
                2       0.85      0.87      0.86       363

         accuracy                           0.86      1400
        macro avg       0.78      0.80      0.79      1400
     weighted avg       0.87      0.86      0.86      1400
```

Figura 22: Generación de predicciones y validación de BETO (Track 1).

La metodología utilizada para entrenar y evaluar **RoBERTuito** es similar a la de BETO salvo algunas diferencias. En primer lugar, la capacidad de procesamiento de textos de RoBERTuito no está plenamente integrada en Hugging Face y por lo tanto es necesario utilizar la librería pysentimiento [38] previamente para procesar los tweets. De esta forma, ya en el *notebook* inicial de preparación de datos TFG_EDA.ipynb se prepararon los subconjuntos de entrenamiento y validación mediante el *split* y la aplicación del método preprocess_tweet de pysentimiento.

```
[ ]  from pysentimiento.preprocessing import preprocess_tweet

[ ]  train_texts, val_texts, train_labels, val_labels = train_test_split(list(map(preprocess_tweet, list(task1_df['tweets']))),
                                                                         list(le.transform(task1_df['label'])), test_size=.2, shuffle=True, random_state=SEED)

[ ]  pd.DataFrame({'train_texts': train_texts, 'train_labels': train_labels}).to_csv('./train_task1_robertuito.csv')
     pd.DataFrame({'val_texts': val_texts, 'val_labels': val_labels}).to_csv('./val_task1_robertuito.csv')
```

Figura 23: Preparación y split de datos para RoBERTuito (Track 1).

De esta forma, en el notebook de RoBERTuito, TFG_Robertuito_Track1.ipynb cargamos directamente los ficheros csv de los subconjuntos de entrenamiento y validación.

```
train_df = pd.read_csv(io.BytesIO(uploaded['train_task1_robertuito.csv']))
val_df = pd.read_csv(io.BytesIO(uploaded['val_task1_robertuito.csv']))

[ ]  train_df
```

	Unnamed: 0	train_texts	train_labels
0	0	¿Quién es Mariquita Pérez? @usuario le cuenta ...	1
1	1	@usuario Si emoji luna nueva con cara emoji m...	2
2	2	Cuando le preguntó a mi papá, qué haría si fue...	0
3	3	"como si es joto" SLAY QUEEN url	0
4	4	LOL. ¿Les había dicho que todas las reglas par...	0
...	...	...	...
5595	5595	Y así fue la gira de trabajo el día de hoy a l...	1
5596	5596	@usuario Nono, yo SI quiero ser abiertamente t...	0
5597	5597	Que puta hueva tengobde entrenar, pero aqui es...	1
5598	5598	Se me ocurrió preguntarle a mi ex marido si él...	1
5599	5599	El 22 de junio habrá marcha del Orgullo LGBT, ...	0

5600 rows × 3 columns

```
[ ]  train_texts, val_texts, train_labels, val_labels = list(train_df['train_texts']), list(val_df['val_texts']), list(train_df['train_labels']), list(val_df['val_labels'])
```

Figura 24: Carga de datos para RoBERTuito (Track 1).

La segunda diferencia en el proceso de entrenamiento de RoBERTuito con respecto a BETO es claramente la necesidad de descargar su propio *checkpoint*. En este caso se utilizó robertuito-base-uncased [40]. La tercera diferencia la define la tasa de aprendizaje, que en este caso comienza en $2e^{-5}$ y finaliza en 0. La última diferencia, y aquella que hace única al proceso de entrenamiento de RoBERTuito en la primera tarea es que, dado que se observó que este modelo tenía una mayor tolerancia a la situación de las clases mal balanceadas (tal y como se explicó en el apartado 2.2. Obtención, exploración y preparación de los datos), se optó por utilizar un conjunto de pesos asociados a cada clase para sesgar el proceso de entrenamiento en favor de la clase minoritaria (P) y en detrimento de la clase mayoritaria (NP). Estos pesos se lograron de forma manual tras múltiples pruebas ajustando el resto de hiperparámetros.

```
[ ]  # Utilizamos un conjunto de pesos que no sesgue en exceso el proceso de entrenamiento.
     class_weight = {0: 0.7, 1: 1, 2: 1.3}

[ ]  class_weight

     {0: 0.7, 1: 1, 2: 1.3}
```

Figura 25: Definición de pesos de clase para RoBERTuito (Track 1).

Al entrenar el modelo añadimos el diccionario de pesos para influir en la importancia de las clases.

```
[ ] model.fit(
        tf_train_dataset,
        validation_data=tf_validation_dataset,
        epochs=num_epochs,
        class_weight=class_weight
    )

    Epoch 1/4
    1400/1400 [==============================] - 412s 259ms/step - loss: 0.4016 - accuracy: 0.8393 - val_loss: 0.3258 - val_accuracy: 0.8779
    Epoch 2/4
    1400/1400 [==============================] - 335s 240ms/step - loss: 0.2139 - accuracy: 0.9170 - val_loss: 0.3775 - val_accuracy: 0.8679
    Epoch 3/4
    1400/1400 [==============================] - 334s 239ms/step - loss: 0.1242 - accuracy: 0.9570 - val_loss: 0.3889 - val_accuracy: 0.8843
    Epoch 4/4
    1400/1400 [==============================] - 335s 240ms/step - loss: 0.0707 - accuracy: 0.9762 - val_loss: 0.4282 - val_accuracy: 0.8779
    <keras.callbacks.History at 0x7fa668998970>
```

Figura 26: Entrenamiento de RoBERTuito (Track 1).

De nuevo, la evaluación del modelo se realiza comparando las predicciones sobre el subconjunto de validación con las clases originales de dicho subconjunto. Obtenemos así las métricas ponderadas *F1-Score* (0.88), *Precision* (0.88) y *Recall* (0.88), junto con el *Accuracy* (0.88) (Figura 27).

```
preds = model.predict(tf_validation_dataset)["logits"]
class_preds = np.argmax(preds, axis=1)
```

```
350/350 [==============================] - 38s 69ms/step
```

```
print(classification_report(class_preds, val_encodings['labels']))
```

```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91       853
           1       0.91      0.88      0.90       386
           2       0.67      0.65      0.66       161

    accuracy                           0.88      1400
   macro avg       0.83      0.82      0.82      1400
weighted avg       0.88      0.88      0.88      1400
```

Figura 27: Generación de predicciones y validación de RoBERTuito (Track 1).

Finalmente, para el entrenamiento y validación de **mDeBERTaV3,** trabajado en el *notebook* TFG mdeberta Track1.ipynb, seguimos los mismos pasos que con BETO, pero en este caso descargamos el *checkpoint* mdeberta-v3-base desde Hugging Face [41]. Con respecto a los hiperparámetros, la arquitectura DeBERTa se recomienda ajustar con un *batch size* mayor (32) y lo hacemos durante 5 épocas. La tasa de aprendizaje inicial es de $5e^{-5}$ y la final de 0. Calculamos las predicciones y generamos el reporte de métricas como en el resto de modelos y obtenemos un *F1-Score* ponderado de 0.83, un *Precision* ponderado de 0.83 y *Recall* ponderado de 0.84, además de un *Accuracy* de 0.84 (Figura 28).

```
preds = model.predict(tf_validation_dataset)["logits"]
class_preds = np.argmax(preds, axis=1)
```

```
44/44 [==============================] - 11s 127ms/step
```

```
print(classification_report(class_preds, val_encodings['labels']))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       872
           1       0.59      0.53      0.56       174
           2       0.83      0.88      0.85       354

    accuracy                           0.84      1400
   macro avg       0.77      0.76      0.77      1400
weighted avg       0.83      0.84      0.83      1400
```

Figura 28: Generación de predicciones y validación de mDeBERTaV3 (Track 1).

2.4.2. Track 2

En esta tarea, el objetivo es entrenar y validar cada uno de los modelos seleccionados sobre un problema multi-etiqueta, donde los inputs son una matriz TF-IDF en el caso de los modelos umbral o directamente los tweets en el caso de los modelos Transformer, y los outputs son 5 posibles etiquetas, que pueden tomar un valor positivo (1) o negativo (0). Cabe destacar que en el caso de la clasificación multi-etiqueta, el método classification_report de sklearn no muestra directamente la métrica *Accuracy*, por lo que para cada modelo habrá que calcularla de forma adicional mediante la función accuracy_score de sklearn.

A. Modelos umbral

Los modelos umbral ajustados a la segunda tarea se encuentran de nuevo en el notebook TFG_EDA.ipynb. Al igual que en la primera tarea, el modelo **Multinomial Naive Bayes** se implementa a través de la clase MultinomialNB de la librería sklearn sobre la matriz TF-IDF de entrenamiento, aunque en este caso es necesario ajustar dicho modelo sobre un clasificador multiobjetivo mediante la clase MultiOutputClassifier de sklearn. Después se utiliza el modelo entrenado para calcular las predicciones sobre la matriz de TF-IDF de validación y finalmente podemos comparar estas predicciones con las clases originales para obtener las métricas.

En la Figura 29 encontramos la implementación del clasificador bayesiano y sus métricas *macro-average F1-Score* de 0.18, *Precision* de 0.20 y *Recall* de 0.17 y un *Accuracy* de 0.78.

```
naive_bayes2 = MultinomialNB()
multilabel_classifier = MultiOutputClassifier(naive_bayes2, n_jobs=-1)
multilabel_classifier.fit(Train_X_Tfidf2, y_train2)
predictions_NB2 = multilabel_classifier.predict(Test_X_Tfidf2)
```

```
[12] print("Naive Bayes Accuracy Score:", round(accuracy_score(predictions_NB2, y_test2), 2))

Naive Bayes Accuracy Score: 0.78
```

```
[13] print(classification_report(predictions_NB2, y_test2))

                precision    recall  f1-score   support

             0       1.00      0.85      0.92       130
             1       0.00      0.00      0.00         0
             2       0.00      0.00      0.00         0
             3       0.00      0.00      0.00         0
             4       0.00      0.00      0.00         0

     micro avg       0.77      0.85      0.81       130
     macro avg       0.20      0.17      0.18       130
  weighted avg       1.00      0.85      0.92       130
   samples avg       0.81      0.85      0.82       130
```

Figura 29: Implementación y reporte de métricas Naive Bayes (Track 2).

Al igual que con Naive Bayes, el modelo **Linear SVC** se implementa a través de la clase SVC de la librería sklearn sobre la matriz TF-IDF de entrenamiento a través del clasificador multi objetivo MultiOutputClassifier. Después se utiliza el modelo entrenado para calcular las predicciones sobre la matriz de TF-IDF de validación y finalmente podemos comparar estas predicciones con las clases originales para obtener de nuevo el reporte de métricas.

En la Figura 29 encontramos la implementación de Linear SVC y sus métricas *macro-average F1-Score* de 0.49, *Precision* de 0.44 y *Recall* de 0.59 y un *Accuracy* de 0.87.

```
SVM2 = SVC(C=1.0, kernel='linear', gamma='auto')
multilabel_classifier = MultiOutputClassifier(SVM2, n_jobs=-1)
multilabel_classifier.fit(Train_X_Tfidf2, y_train2)
predictions_SVM2 = multilabel_classifier.predict(Test_X_Tfidf2)
```

```
[15] print("SVM Accuracy Score:", round(accuracy_score(predictions_SVM2, y_test2), 2))

SVM Accuracy Score: 0.87
```

```
[16] print(classification_report(predictions_SVM2, y_test2))
```

	precision	recall	f1-score	support
0	1.00	0.93	0.96	118
1	0.67	1.00	0.80	6
2	0.00	0.00	0.00	0
3	0.54	1.00	0.70	7
4	0.00	0.00	0.00	0
micro avg	0.87	0.94	0.90	131
macro avg	0.44	0.59	0.49	131
weighted avg	0.96	0.94	0.94	131
samples avg	0.91	0.93	0.91	131

Figura 30: Implementación y reporte de métricas Linear SVC (Track 2).

B. Modelos Transformer

La metodología seguida para la implementación de modelos Transformer en la segunda tarea es muy similar a la de la primera tarea. Cada modelo se desarrolló en un *notebook* independiente y se utilizaron las mismas herramientas y la misma secuencia de instrucciones, además los mismos modelos fueron utilizados. La diferencia fundamental radica en la preparación de los subconjuntos de entrenamiento y validación, que en el caso de la segunda la variable output cuenta con 5 valores, uno por cada etiqueta. De esta forma, en el proceso de entrenamiento la función de pérdida que se debe utilizar también debe ser distinta, ya que ahora estaríamos trabajando sobre un problema de clasificación binaria para cada una de las etiquetas.

Los hiperparámetros y parámetros utilizados son los mismos que en la primera tarea, aunque en este caso nunca se implementó un conjunto de pesos para sesgar el proceso de entrenamiento en favor de una clase mayoritaria como ocurría anteriormente en el caso de RoBERTuito, ya que en esta tarea cada etiqueta consiste en un problema de

clasificación binaria. Igualmente los hiperparámetros fueron determinados tras un proceso intensivo de prueba y error manual. Se muestran los hiperparámetros ajustados para los modelos en esta tarea:

Model	Epochs	Batch size	start Learning rate	end Learning rate	dropout
BETO	4	4	$5e^{-5}$	0	0.2
RoBERTuito	7	4	$2e^{-5}$	0	-
mDeBERTaV3	5	32	$5e^{-5}$	0	-

Figura 31: Selección de hiperparámetros para modelos Transformer (Track 2).

Comenzando por el modelo **BETO** (notebook TFG_BETO_Track2.ipynb), realizamos una carga limpia de los datos de entrenamiento a partir del fichero csv original de la segunda tarea en un Pandas DataFrame. Antes de poder obtener los subconjuntos de entrenamiento y validación necesitamos comprimir todas las variables *output* en una única columna de listas que incluyan los valores binarios de las etiquetas. Como siempre, para asegurarnos que estos subconjuntos son siempre los mismos, utilizamos la misma semilla al realizar el *split*.

```
task2_df = pd.read_csv(io.BytesIO(uploaded['multi_train_labels.csv']))
```

```
label_cols = ['G','L','B','T','O']
task2_df['labels'] = list(task2_df[label_cols].values)
```

```
task2_df
```

	tuit	G	L	B	T	O	labels
0	—Bueno, pues ya pasó la euforia papá, ahora sí...	1	0	0	0	0	[1, 0, 0, 0, 0]
1	Esos de BTS se me hacen seres asexuales, eunuc...	0	0	0	0	1	[0, 0, 0, 0, 1]
2	La partida de skywars que me ha hecho enojar m...	1	0	0	0	0	[1, 0, 0, 0, 0]
3	Se te acabo tu juego maldito joto asqueroso, e...	1	0	0	0	0	[1, 0, 0, 0, 0]
4	@Lunaazulada_ Será a los jotitos.??ellos prefi...	1	0	0	0	0	[1, 0, 0, 0, 0]
...	...	...	...	...	...	...	...
857	@alf_avila_ @jrisco @penileyramirez De mi part...	1	0	0	0	0	[1, 0, 0, 0, 0]
858	@CxavezE @altazor08 Se acalambra el putito 🐓...	1	0	0	0	0	[1, 0, 0, 0, 0]
859	Psst, solo les quiero spoilerear la platica qu...	0	0	0	0	1	[0, 0, 0, 0, 1]
860	@navirri8 @DemonCerda @MARIOHDZRUIZ ps no somo...	1	0	0	0	1	[1, 0, 0, 0, 1]
861	Alguien es bien nena, mariquita y flojo y adem...	1	0	0	0	0	[1, 0, 0, 0, 0]

862 rows × 7 columns

```
train_texts, val_texts, train_labels, val_labels = train_test_split(list(task2_df['tuit']), task2_df['labels'],
                                                                    test_size=.15, random_state=SEED, shuffle=True)
```

Figura 32: Carga y split de datos para BETO (Track 2).

A continuación, procedemos como en la primera tarea, descargando el *checkpoint* bert-base-spanish-wwm-cased y construyendo los tokenizadores que nos permiten crear los datasets de Tensorflow para el entrenamiento y validación. Con respecto a los hiperparámetros, elegimos un *batch size* de 4 durante 4 épocas y una tasa de aprendizaje

inicial de $5e^{-5}$. En el caso de los modelos orientados a resolver problemas de clasificación multi-etiqueta, cuando se instancia el modelo mediante la clase TFAutoModelForSequenceClassification, es necesario especificar el tipo de problema. Además utilizamos una probabilidad de dropout del 20%.

```
model = TFAutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=5, problem_type="multi_label_classification",
                                                             attention_probs_dropout_prob=0.1,
                                                             hidden_dropout_prob=0.2,
                                                             from_pt=True)
```

Figura 33: Creación del modelo multi-etiqueta para BETO (Track 2).

Finalmente podemos calcular las métricas en base a las predicciones sobre el subconjunto de validación. De esta forma obtenemos unas métricas *macro-average F1-Score* de 0.52, *Precision* de 0.51 y *Recall* de 0.59 y un *Accuracy* de 0.86 (Figura 34).

```
preds = model.predict(tf_validation_dataset)["logits"]
class_preds = [(r > 0).astype(np.int) for r in preds]
```

```
[ ] print("Beto Accuracy Score:", round(accuracy_score(class_preds, list(val_encodings['labels'])), 2))

    Beto Accuracy Score: 0.86
```

```
[ ] print(classification_report(class_preds, list(val_encodings['labels'])))
```

	precision	recall	f1-score	support
0	0.96	0.97	0.97	109
1	0.78	0.58	0.67	12
2	0.00	0.00	0.00	0
3	0.69	0.90	0.78	10
4	0.12	0.50	0.20	2
micro avg	0.87	0.92	0.89	133
macro avg	0.51	0.59	0.52	133
weighted avg	0.91	0.92	0.92	133
samples avg	0.91	0.93	0.92	133

Figura 34: Generación de predicciones y validación de BETO (Track 2).

Al igual que ocurría en la primera tarea, para entrenar y evaluar **RoBERTuito** (*notebook* TFG_Robertuito_Track2.ipynb) primero es necesario utilizar la librería pysentimiento para procesar los tweets. De esta forma, ya en el *notebook* inicial de preparación de datos TFG_EDA.ipynb se prepararon los subconjuntos de entrenamiento y validación también para el Track 2 orientado a este modelo.

Con respecto a los hiperparámetros, en el caso de RoBERTuito la tasa de aprendizaje comienza en $2e^{-5}$ y finaliza en 0, el *batch size* de 4 y entrenamos durante 7 épocas. Recordamos de nuevo que para esta tarea no se implementó un sistema de pesos por clase para influir en la fase de entrenamiento. Se calculan las métricas sobre las predicciones obteniendo unas métricas *macro-average F1-Score* de 0.52, *Precision* de 0.48 y *Recall* de 0.58 y un *Accuracy* de 0.88 (Figura 34).

```
preds = model.predict(tf_validation_dataset)["logits"]
class_preds = [(r > 0).astype(np.int) for r in preds]
```

```
[ ] print("Robertuito Accuracy Score:", round(accuracy_score(class_preds, list(val_encodings['labels']))*100, 2))

    Robertuito Accuracy Score: 87.69
```

```
[ ] print(classification_report(class_preds, list(val_encodings['labels'])))

                precision    recall  f1-score   support

             0       0.96      0.98      0.97       108
             1       0.67      1.00      0.80         6
             2       0.00      0.00      0.00         0
             3       0.77      0.91      0.83        11
             4       0.00      0.00      0.00         0

     micro avg       0.86      0.98      0.91       125
     macro avg       0.48      0.58      0.52       125
  weighted avg       0.93      0.98      0.95       125
   samples avg       0.91      0.94      0.92       125
```

Figura 35: Generación de predicciones y validación de RoBERTuito (Track 2).

Finalmente, para el entrenamiento y validación de **mDeBERTaV3** (notebook TFG_mdeberta_Track2.ipynb) seguimos los mismos pasos que con BETO al igual que ocurría en la primera tarea, pero utilizando el *checkpoint* mdeberta-v3-base. Con respecto a los hiperparámetros, entrenamos con un *batch size* de 32 durante 5 épocas. La tasa de aprendizaje inicial es de $5e^{-5}$ y la final de 0. Calculamos las predicciones y generamos el reporte de métricas como hicimos con el resto de modelos, obteniendo un *macro-average F1-Score* de 0.19, *macro-average Precision* de 0.20 y *macro-average Recall* de 0.19, junto con un *Accuracy* de 0.78

```
preds = model.predict(tf_validation_dataset)["logits"]
class_preds = [(r > 0).astype(np.int) for r in preds]
```

```
[ ] print("mdeberta Accuracy Score:", round(accuracy_score(class_preds, list(val_encodings['labels'])), 2))

    mdeberta Accuracy Score: 0.78
```

```
[ ] print(classification_report(class_preds, list(val_encodings['labels'])))

                precision    recall  f1-score   support

             0       0.98      0.95      0.96       114
             1       0.00      0.00      0.00         0
             2       0.00      0.00      0.00         0
             3       0.00      0.00      0.00         0
             4       0.00      0.00      0.00         0

     micro avg       0.76      0.95      0.84       114
     macro avg       0.20      0.19      0.19       114
  weighted avg       0.98      0.95      0.96       114
   samples avg       0.80      0.83      0.81       114
```

Figura 36: Generación de predicciones y validación de mDeBERTaV3 (Track 2).

En el apartado 3.1. Resultados analíticos se discuten las métricas obtenidas en los distintos modelos y se comparan en profundidad.

3. Resultados

En este apartado se recopilan en primer lugar, los resultados obtenidos de la implementación de los modelos explicados anteriormente, así como una serie de conclusiones relacionadas con la comparación entre modelos, en segundo lugar los resultados obtenidos en la competición HOMO-MEX 2023 publicados por los/las organizadores/as y por último se exponen en detalle los distintos productos obtenidos fruto de este proyecto.

3.1. Resultados analíticos

Los resultados obtenidos en el apartado 2.4. Experimentación se puede observar a continuación, expuestos de la misma forma que en la contribución científica enviada a la tarea HOMO-MEX. Para cada una de las tareas, se muestra cada modelo con las métricas seleccionadas. Se resaltan los resultados del modelo RoBERTuito ya que este logró los mejores resultados en la competición, tal y como se expondrá en el apartado 3.2. Resultados de la competición.

Recordamos que en la primera tarea (Track 1) se seleccionaron las métricas *Accuracy* y media ponderada para *F1-Score*, *Precision* y *Recall*, mientras que para la segunda tarea (Track 2) se seleccionaron las métricas *Accuracy* y media macro para *F1-Score*, *Precision* y *Recall*. En la Figura 37 se encuentran los resultados de todos los modelos implementados en la primera tarea (Track 1) de acuerdo a sus métricas, en la Figura 38 encontramos los resultados para la segunda tarea (Track 2).

Model	w.F1-Score	w.Precision	w.Recall	Accuracy
Multinomial Naive Bayes	0.76	0.94	0.68	0.68
Linear SVC	0.84	0.85	0.83	0.83
BETO	0.86	0.87	0.86	0.86
RoBERTuito	**0.88**	**0.88**	**0.88**	**0.88**
mDeBERTaV3	0.83	0.83	0.84	0.84

Figura 37: Resultados por *weighted-average F1-Score* por modelo.

Model	macro F1-Score	macro Precision	macro Recall	Accuracy
Multinomial Naive Bayes	0.18	0.20	0.17	0.78
Linear SVC	0.49	0.44	0.59	0.87
BETO	0.52	0.51	0.59	0.86
RoBERTuito	**0.52**	**0.48**	**0.58**	**0.88**
mDeBERTaV3	0.19	0.20	0.19	0.78

Figura 38: Resultados por *macro-average F1-Score* por modelo.

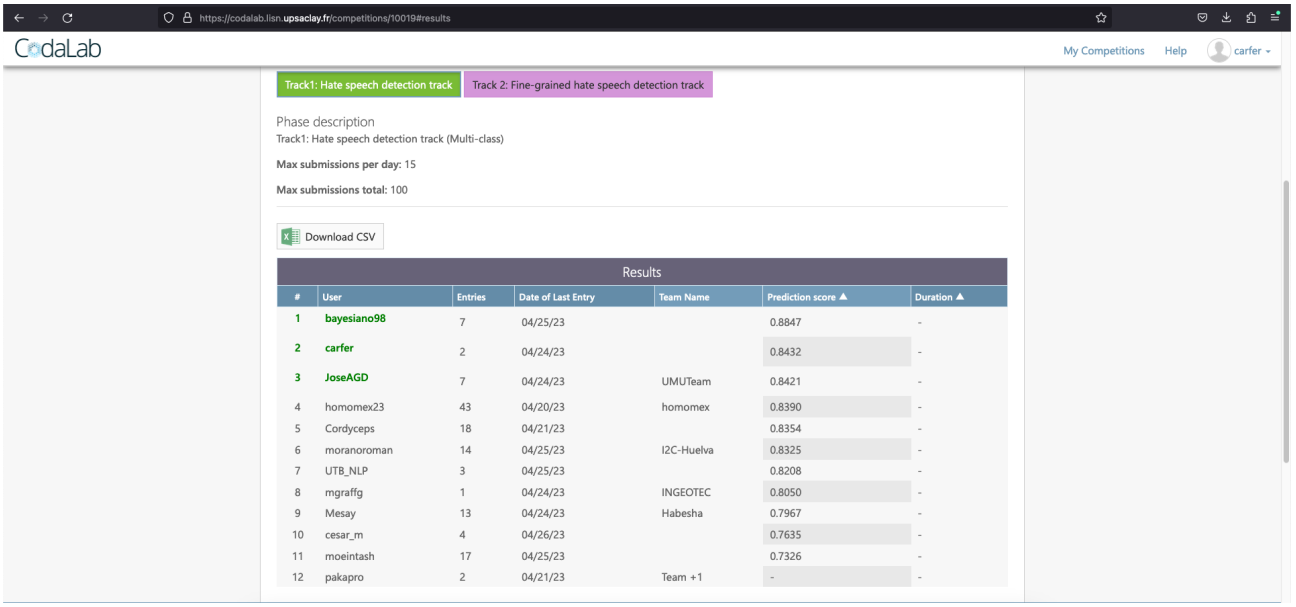
Las principales observaciones que podemos extraer a la hora de comparar los resultados entre los modelos son:

1. Los modelos RoBERTuito o BETO obtienen los mejores resultados en todas las métricas en ambas tareas. Recordamos que estos son modelos Transformer pre entrenados únicamente en español.
2. El modelo mDeBERTaV3, que es aquel que cuenta con la arquitectura más moderna y avanzada, no logra mejorar las métricas de BETO y RoBERTuito en la primera tarea, y en el caso de la segunda tarea obtiene unos resultados casi tan bajos como el modelo que peor rinde, que es Multinomial Naive Bayes (Figura 38). Recordamos que mDeBERTaV3 fue pre entrenado en múltiples idiomas.
3. El modelo Linear SVC logra unos resultados excelentes en ambas tareas, siendo un modelo mucho más sencillo de implementar y con un coste computacional considerablemente más bajo.

Estas observaciones nos permiten extraer una conclusión sorprendente y con implicaciones de cara a la implementación real de un sistema de clasificación de textos como el investigado en forma de producto. Dentro del contexto experimental planteado en este proyecto, podemos concluir que un modelo tan simple como Linear SVC, bien ajustado, rinde casi al nivel de modelos Transformer porque en el contexto de textos cortos en redes sociales, la existencia de términos LGTBI-fóbicos y su tipología son las características más importantes en la tarea de clasificación planteada, mientras que el orden de las palabras o la similitud semántica entre términos, que son características que los modelos Transformer son capaces de capturar, no tienen tanta relevancia. Si quisiéramos utilizar un clasificador de estas características para filtrar contenido LGTBI-fobico en una red social, habría que tener en cuenta el coste computacional en tiempo de entrenamiento e inferencia para hacer de la aplicación un sistema escalable y sostenible, teniendo que comparar las ventajas y desventajas de implementar modelos tradicionales como el clasificador lineal SVC frente a modelos Transformer.

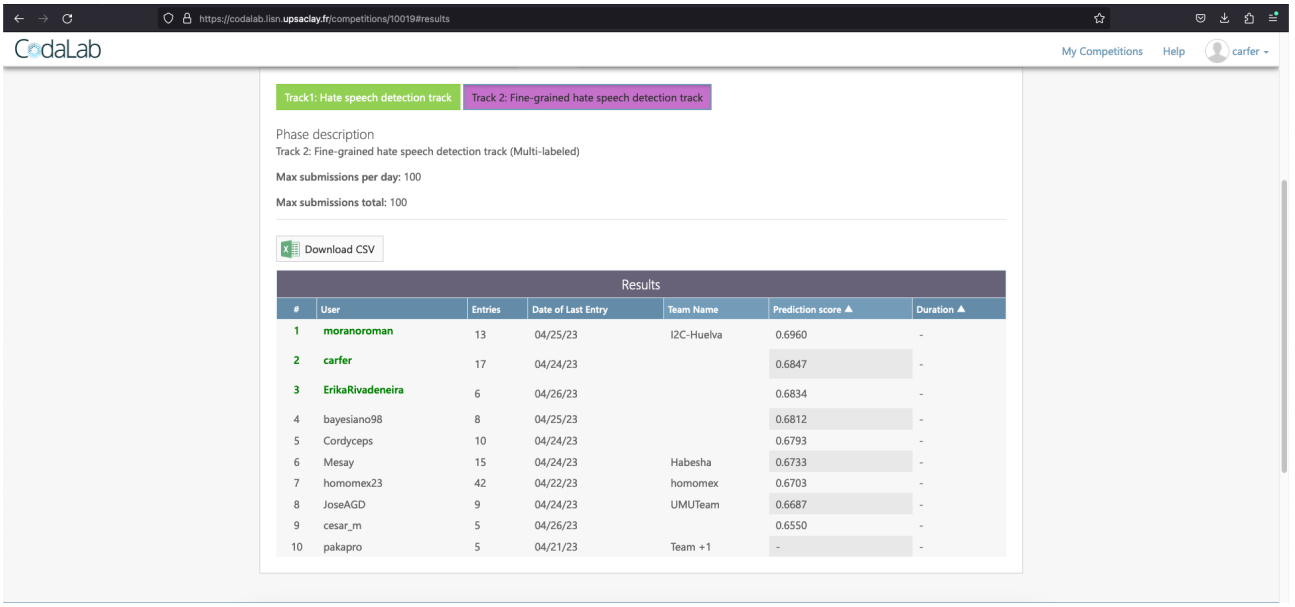
3.2. Resultados de la competición

Como parte de la tarea HOMO-MEX organizada por IberLEF 2023, se desarrolló una fase de evaluación de los modelos de los participantes en la que se ofrecieron conjuntos de evaluación para cada una de las tareas (Track 1 y Track 2). Estos conjuntos no estaban etiquetados, de tal manera que el objetivo era aplicar los modelos desarrollados con el objetivo de generar etiquetas que los/las organizadores/as pudieran validar. Para ambas tareas, todos los modelos basados en arquitecturas Transformer fueron utilizados para la generación de etiquetas, y en el caso de RoBERTuito, se logró la segunda posición en la clasificación global de la competición en ambas tareas, logrando unas métricas *F1-Score* 0.84 en Track 1 (Figura 39) y *macro-average F1-Score* en Track 2 (Figura 40). Los resultados fueron enviados a través del usuario “carfer”. Cabe destacar que los modelos umbral no fueron utilizados en este proceso de evaluación.



#	User	Entries	Date of Last Entry	Team Name	Prediction score ▲	Duration ▲
1	bayesiano98	7	04/25/23		0.8847	-
2	carfer	2	04/24/23		0.8432	-
3	JoseAGD	7	04/24/23	UMUTeam	0.8421	-
4	homomex23	43	04/20/23	homomex	0.8390	-
5	Cordyceps	18	04/21/23		0.8354	-
6	moranoroman	14	04/25/23	I2C-Huelva	0.8325	-
7	UTB_NLP	3	04/25/23		0.8208	-
8	mgraffg	1	04/24/23	INGEOTEC	0.8050	-
9	Mesay	13	04/24/23	Habesha	0.7967	-
10	cesar_m	4	04/26/23		0.7635	-
11	moeintash	17	04/25/23		0.7326	-
12	pakapro	2	04/21/23	Team +1	-	-

Figura 39: Leaderboard con resultados oficiales de Track 1.



#	User	Entries	Date of Last Entry	Team Name	Prediction score ▲	Duration ▲
1	moranoroman	13	04/25/23	I2C-Huelva	0.6960	-
2	carfer	17	04/24/23		0.6847	-
3	ErikaRivadeneira	6	04/26/23		0.6834	-
4	bayesiano98	8	04/25/23		0.6812	-
5	Cordyceps	10	04/24/23		0.6793	-
6	Mesay	15	04/24/23	Habesha	0.6733	-
7	homomex23	42	04/22/23	homomex	0.6703	-
8	JoseAGD	9	04/24/23	UMUTeam	0.6687	-
9	cesar_m	5	04/26/23		0.6550	-
10	pakapro	5	04/21/23	Team +1	-	-

Figura 40: Leaderboard con resultados oficiales de Track 2.

3.3. Productos obtenidos

Tal y como se introducía en el apartado 1.6. Breve sumario de productos obtenidos, como resultado de este proyecto se obtuvieron dos productos. En este apartado se explicarán en detalle.

3.3.1. Paper académico enviado a IberLEF 2023

Como parte final de la tarea HOMO-MEX 2023, se escribió un paper académico en formato de *working notes* junto con la tutora del TFG Montse Cuadros. Este paper, *Hate Speech Detection Against the Mexican Spanish LGBTQ+ Community Using BERT-based Transformers*, explica en inglés la tarea realizada, los modelos implementados, los experimentos realizados y las conclusiones obtenidas, siguiendo la metodología CEUR Workshop Proceedings [42], que es el formato que pidieron en la conferencia donde se ubica la tarea. A parte de las conclusiones relacionadas con las diferencias observadas en el rendimiento entre métodos clásicos y modelos basados en Transformers como las exploradas en esta memoria, en el paper se hace también énfasis en los buenos resultados obtenidos en la fase de evaluación sobre los conjuntos de datos ofrecidos por los/las organizadores/as de la tarea, tal y como se explica en el apartado 3.2. Resultados de la competición. La actual versión enviada a IberLEF puede encontrarse en el Anexo 1. El día 16 de junio de 2023 se recibieron las revisiones realizadas por los/las organizadores/as de HOMO-MEX, siendo estas bastante positivas. Para la versión final del paper se deberán realizar pequeñas modificaciones en base a estas correcciones.

3.3.2. Repositorio Github

El segundo producto obtenido es un repositorio de Github [17] en el que se puede encontrar el código que define todos los pasos realizados en el desarrollo del proyecto. Tal y como se explicó en detalle en el apartado 2.4. Experimentación, cada modelo utilizado para cada una de las tareas fue implementado de forma independiente en un *notebook* de Python que cuenta con su nombre. Además la implementación de los modelos clásicos y la fase de preparación y limpieza de datos se incluyen también en su propio *notebook*. Adicionalmente, se incluye en el repositorio un directorio con todos los ficheros utilizados, es decir, tanto los ficheros iniciales como aquellas transformaciones realizadas previas al entrenamiento de los modelos.

4. Conclusiones y trabajos futuros

Considerando los resultados obtenidos durante el desarrollo del proyecto y la creación de los productos generados, las principales conclusiones que se obtienen son:

1. Los modelos basados en arquitecturas BERT y pre entrenados en español logran unos resultados excelentes a la hora de detectar mensajes de odio contra la comunidad LGTBI+ mexicana hispanohablante, tal y como se extrae de los resultados obtenidos tanto en los procesos internos de validación (3.1. Resultados Analíticos) como en la evaluación realizada por la organización de HOMO-MEX (3.2. Resultados de la competición). Cabe destacar que en la clasificación global de la tarea se logró un segundo puesto en ambos *tracks*.
2. Los modelos estadísticos clásicos de minería de textos como Linear SVC pueden lograr unos resultados a priori casi tan buenos como los de los modelos basados en arquitecturas

BERT si son ajustados sobre matrices TF-IDF y sus parámetros están bien definidos, tal y como demuestran los experimentos realizados internamente (3.1. Resultados Analíticos).

3. Dada la ambigüedad que puede presentar una implementación de un sistema de detección de mensajes de odio en el contexto de redes sociales por las conclusiones 1 y 2, a la hora de decidir entre seleccionar entre un modelo Transformer y un modelo clásico, habrá que tener en cuenta variables relacionadas con la facilidad de implementación, el coste energético o el coste de despliegue.

En el contexto de la evaluación de los modelos realizada sobre los subconjuntos de validación, obtenidos a través de la división del conjunto de datos original y no a través del conjunto de datos de evaluación aportado por los/las organizadores/as, las conclusiones resultan sorprendentes, ya que al comienzo de este proyecto no se preveía que las métricas de un modelo tradicional como Linear SVC sobre matriz TF-IDF fuesen a lograr valores tan similares a los de modelos basados en arquitecturas BERT.

A pesar de haberse implementado de forma satisfactoria un conjunto de modelos avanzados para la clasificación de textos de tipo *encoder-only* (BETO, RoBERTuito, mDeBERTaV3), al comienzo del proyecto se planteó la intención de implementar otras arquitecturas Transformer de tipo *decoder-only*, como los populares modelos generativos (GPT-3). Esto no se llevó a cabo por dos motivos: los modelos de tipo BERT ya lograron muy buenos resultados en la clasificación de la tarea HOMO-MEX y el ajuste de los modelos generativos para tareas de clasificación de textos planteó la necesidad de un periodo de tiempo adicional superior al que se tenía tras el entrenamiento de los primeros.

Con respecto a los impactos definidos en el apartado 1.3. Impacto en sostenibilidad, ético-social y de diversidad, encontramos las siguientes conclusiones por dimensión:

- Dimensión sostenibilidad: se conocen las diferencias significativas en costes computacionales entre modelos entrenables mediante CPU como Linear SVC sobre matrices TF-IDF y modelos que requieren de GPU como RoBERTuito. Tener en cuenta estas diferencias es imprescindible para impulsar la sostenibilidad en los sistemas de clasificación de textos.
- Dimensión ético-social: dado que es la organización proveedora de los datos la que gestiona los conjuntos utilizados para el entrenamiento y la validación de los modelos, no se ha trabajado el posible impacto negativo relacionado con la privacidad explicado en el apartado 1.3.
- Dimensión diversidad: el TFG ha contribuido a su objetivo fundamental de contribuir en la búsqueda de la igualdad para las personas del colectivo LGTBI+ a través de la detección de contenido LGTBI-fóbico en el contexto de las redes sociales, lo cual se alinea con lo planteado en el diseño de los impactos establecidos en el apartado 1.3.

Con respecto a las líneas de trabajo futuro, se exploran tres posibilidades:

1. Desarrollo de un análisis causal de la efectividad de los modelos según sus características. Se ha demostrado en el entorno experimental que el modelo Linear SVC es capaz de rendir casi al nivel de los modelos Transformer por el formato de texto con el que ha trabajado: mensajes cortos en el contexto de redes sociales. Convendría explorar en profundidad las razones específicas de esta similitud de resultados.
2. Implementación de modelos generativos para la clasificación de textos, ya que convendría comprobar cómo estos modelos tan populares y extendidos se comparan con las arquitecturas BERT y con los modelos clásicos.
3. Diseño y desarrollo de una aplicación web para el análisis en tiempo real de textos cortos de una red social para la detección de contenido LGTBI-fóbico. Aunque esta línea se sale de la disciplina del Procesamiento del Lenguaje Natural puro, el objetivo del desarrollo de modelos para la mejora de las condiciones de colectivos oprimidos debe ser siempre práctica, y la puesta en marcha de una herramienta orientada a la utilización de estos modelos podría contribuir de forma positiva al aumento de la visibilidad de los problemas que sufre el colectivo en las redes sociales.

5. Glosario

Accuracy: porcentaje de muestras correctamente clasificadas/etiquetadas.

BERT: Bidirectional Encoder Representations from Transformers.

Checkpoint: punto de control de un modelo de machine learning, generalmente disponible para su descarga.

F1-Score: media armónica de las métricas Precision y Recall.

IberLEF: Iberian Languages Evaluation Forum.

Modelo umbral: modelo estadístico o de aprendizaje automático utilizado como punto inicial de evaluación y sobre el que se trata de mejorar los resultados.

Precision: habilidad del clasificador de no etiquetar muestras negativas como positivas.

Recall: habilidad del clasificador de encontrar todas las muestras positivas.

Transformer: modelo de red neuronal artificial que adquiere conocimiento contextual observando las relaciones entre los elementos de una serie de datos secuencial, como por ejemplo las palabras dentro de una frase.

6. Bibliografía

[1] GLAAD. (2021). 2021 Social Media Safety Index. https://assets.glaad.org/m/7adb1180448da194/original/Social-Media-Safety-Index-2023.pdf

[2] Associated Press. (2023, June 15). Twitter is the 'most dangerous' social media platform for LGBTQ users, GLAAD says. NBC News. https://www.nbcnews.com/nbc-out/out-news/twitter-dangerous-social-media-platform-lgbtq-users-glaad-says-rcna89530

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[4] OpenAI. (2022, November 30). Introducing ChatGPT. OpenAI. Retrieved June 17, 2023, from https://openai.com/blog/chatgpt

[5] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., ... & Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. Nature, 596(7873), 583-589.

[6] THE GLOBAL GOALS. (n.d.). Goal 9: Industry, innovation and infrastructure. The Global Goals. Retrieved June 17, 2023, from https://www.globalgoals.org/goals/9-industry-innovation-and-infrastructure/

[7] THE GLOBAL GOALS. (n.d.). Goal 10: Reduced inequalities. The Global Goals. Retrieved June 17, 2023, from https://www.globalgoals.org/goals/10-reduced-inequalities/

[8] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. & others (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877--1901.

[9] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (n.d.). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 1, 4171-4186.

[10] G. Bel-Enguix, H. Gómez-Adorno, G. Sierra, J. Vásquez, S.-T. Andersen, S.-L. Ojeda-Trueba (2023). Overview of HOMO-MEX at IberLEF 2023: Paraphrase Detection in Spanish Shared Task. Procesamiento del Lenguaje Natural, 71.

[11] SEPLN. (n.d.). Iberian Languages Evaluation Forum 2023 – SEPLN 2023. SEPLN 2023. Retrieved June 17, 2023, from http://sepln2023.sepln.org/en/iberlef-en/

[12] HOMO-MEX Competition. (n.d.). CodaLab - Competition. Retrieved June 17, 2023, from https://codalab.lisn.upsaclay.fr/competitions/10019#learn_the_details

[13] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016, November). Tensorflow: a system for large-scale machine learning. In Osdi (Vol. 16, No. 2016, pp. 265-283).

[14] Hugging Face. (2016). Hugging Face. Hugging Face – The AI community building the future. Retrieved June 17, 2023, from https://huggingface.co/

[15] Overleaf. (n.d.). Overleaf, Online LaTeX Editor. Retrieved June 17, 2023, from https://www.overleaf.com/

[16] ProjectLibre - Project Management download. (2023, January 25). SourceForge. Retrieved June 17, 2023, from https://sourceforge.net/projects/projectlibre/

[17] Fernández, C. (2023). tfg_homo-mex repository. Github. Retrieved June 17, 2023, from https://github.com/cfernandezros/tfg_homo-mex

[18] He, P., Gao, J., & Chen, W. (2023). DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing.

[19] scikit-learn developers. (n.d.). 1.12. Multiclass and multioutput algorithms — scikit-learn 1.2.2 documentation. Scikit-learn. Retrieved June 17, 2023, from https://scikit-learn.org/stable/modules/multiclass.html

[20] LGTBI-fobia. (2023, January 6). Retrieved June 17, 2023, from https://dej-enclave.rae.es/lema/lgtbi-fobia

[21] Qué es la LGTBIfobia. (2019, June 1). Observatorio Andaluz contra la Homofobia, Bifobia y Transfobia. Retrieved June 17, 2023, from https://observatorioandaluzlgbt.org/que-es-la-lgtbifobia/

[22] pandas.DataFrame — pandas 2.0.2 documentation. (n.d.). Pandas. Retrieved June 17, 2023, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

[23] NLTK. (2023). NLTK :: Natural Language Toolkit. Retrieved June 17, 2023, from https://www.nltk.org/

[24] Benzahia, L. (n.d.). nltk.stem.snowball module. NLTK. Retrieved June 17, 2023, from https://www.nltk.org/api/nltk.stem.snowball.html

[25] Mueller, A. (2020). WordCloud for Python documentation — wordcloud 1.8.1 documentation. Retrieved June 17, 2023, from https://amueller.github.io/word_cloud/

[26] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

[27] Schutze, H., Manning, C. D., & Raghavan, P. (2008). Introduction to information retrieval. Cambridge University Press. 234-265

[28] Schutze, H., Manning, C. D., & Raghavan, P. (2008). Introduction to information retrieval. Cambridge University Press. 319-325

[29] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., & Pérez, J. (2020). Spanish pre-trained bert model and evaluation data. PML4DC at ICLR 2020.

[30] Pérez, J. M., Furman, D. A., Alonso Alemany, L., & Luque, F. (2022). RoBERTuito: a pre-trained language model for social media text in Spanish. Proceedings of the Thirteenth

Language Resources and Evaluation Conference, 7235-7243. https://aclanthology.org/2022.lrec-1.785

[31] scikit-learn developers. (n.d.). sklearn.naive_bayes.MultinomialNB — scikit-learn 1.2.2 documentation. Scikit-learn. Retrieved June 17, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

[32] scikit-learn developers. (n.d.). sklearn.svm.SVC — scikit-learn 1.2.2 documentation. Scikit-learn. Retrieved June 17, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[33] Datasets documentation. (n.d.). Hugging Face. Retrieved June 17, 2023, from https://huggingface.co/docs/datasets/index

[34] Transformers. (n.d.). Hugging Face. Retrieved June 17, 2023, from https://huggingface.co/docs/transformers/index

[35] Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. arXiv.

[36] dccuchile/bert-base-spanish-wwm-cased · Hugging Face. (n.d.). Hugging Face. Retrieved June 17, 2023, from https://huggingface.co/dccuchile/bert-base-spanish-wwm-cased

[37] tf.keras.losses.SparseCategoricalCrossentropy. (n.d.). TensorFlow. Retrieved June 17, 2023, from https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

[38] Pérez, J. M., Giudici, J. C., & Luque, F. (2021). pysentimiento: A python toolkit for sentiment analysis and socialnlp tasks. arXiv preprint arXiv:2106.09462.

[40] pysentimiento/robertuito-base-uncased · Hugging Face. (n.d.). Hugging Face. Retrieved June 17, 2023, from https://huggingface.co/pysentimiento/robertuito-base-uncased

[41] microsoft/mdeberta-v3-base · Hugging Face. (n.d.). Hugging Face. Retrieved April 17, 2023, from https://huggingface.co/microsoft/mdeberta-v3-base

[42] CEUR Workshop Proceedings. (n.d.). CEUR-WS.org - CEUR Workshop Proceedings (free, open-access publishing, computer science/information systems). Retrieved June 17, 2023, from https://ceur-ws.org/

7. Anexos

7.1. Anexo 1: Paper académico HOMO-MEX 2023

Hate Speech Detection Against the Mexican Spanish LGBTQ+ Community Using BERT-based Transformers

Carlos Fernández Rosauro[1,*], Montse Cuadros[2]

[1]Universitat Oberta de Catalunya, Barcelona, Spain
[2]SNLT group at Vicomtech Foundation, Basque Research and Technology Alliance (BRTA), Mikeletegi Pasealekua 57, Donostia/San-Sebastián, 20009, Spain

Abstract

In this paper we present our approach to the HOMO-MEX task: Hate speech detection in Online Messages directed tOwards the MEXican spanish speaking LGBTQ+ population. We present our results for both Track 1: Hate speech detection track, in which the aim is to indicate whether a set of tweets exhibit LGBT+phobic content or not, and Track 2: Fine-grained hate speech detection track (Multi-labeled), in which the tweets labeled as LGBT+phobic need to be classified according to the type of LGBT+phobia they show. We utilized both classical machine learning and Transformer-based deep learning models focused on BERT-like architectures to tackle both tracks. The model that achieved the best results in terms of F1-Score (0.84 in Track 1) and macro-average F1-Score (0.68 in Track 2) was robertuito-base-uncased. With this model our team reached the 2nd position in both tracks.

Keywords

NLP, Text Classification, Hate Speech, Deep Learning

1. Introduction

This paper describes our participation in HOMO-MEX 2023 shared task, which is part of the iberLEF Conference. This challenge is focused on Hate speech detection in Online Messages directed tOwards the MEXican spanish speaking LGBTQ+ population. It is split in two tasks:

- In Track 1, participants have been provided with a collection of tweets and tasked with determining whether these tweets contain LGBT+phobic content. The classification options available for the tweets are: LGBT+phobic (P), not LGBT+phobic (NP), or not LGBT+related (NA).
- Track 2, on the other hand, involves fine-grained hate speech detection. Participants are required to identify the specific type of LGBT+phobia conveyed in the labeled tweets. The classification categories for this track include Lesbophobia (L), Gayphobia (G), Biphobia (B), Transphobia (T), and/or other forms of LGBT+phobia (O). Participants have the flexibility to assign one or more labels to each tweet. For instance, a tweet might be

IberLEF 2023, September 2023, Jaén, Spain
*Corresponding author.
cfernandezrosa@uoc.edu (C. F. Rosauro); mcuadros@vicomtech.org (M. Cuadros)
https://github.com/cfernandezros/ (C. F. Rosauro)
0000-0002-3620-1053 (M. Cuadros)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

assigned multiple labels such as "L", "G", and "B" while another tweet might only receive the label "O".

We refer the reader to the overview article [1] of the HOMO-MEX 2023 competition for further information. Our team has participated in both tasks, implementing baseline models and more sophisticated systems based on BERT-like [2] Transformers. Both tasks are related to text classification tasks where mainly differ in that Track 1 is a multi-class problem and Track 2 is a multi-label problem.

This paper is organized as follows, section 2 presents the datasets available for the shared task which have been used. Section 3 presents the basic system used for both tracks. Section 4 presents the experimentation performed in Track 1 and in Track 2. Finally, section 5 draw some concluding remarks of our participation in the shared task.

2. Dataset

In the shared task, the official dataset for Track 1 regarding training data contains 7000 tweets labelled as either LGBT+phobic (P), not LGBT+phobic (NP) or not LGBT+related (NA) sorted by an index. Regarding Track 2, training dataset contains 862 LGBT+phobic tweets along with 5 columns that define the type of hate speech that the tweet contains in a binary-variable manner (1 or 0). This way, Track 2 tweets can contain one or multiple of the following behaviours: Lesbophobia (L), Gayphobia (G), Biphobia (B), Transphobia (T), and/or other LGBT+phobia (O).

Track 1 training set is split into 80% for training and 20% for validation, while Track 2 training set is split into 85% for training and 15% for validation . In both cases this was done by randomly selecting the validation split using a preset seed. Table 1 shows the number of tweets per track in train and validation sets and the average word tokens per tweet.

Table 1
Training dataset for Track 1 and Track 2 in number of tweets and avg. word tokens per tweet

	Track 1	Track 2
Train	5600	1400
Validation	732	132
Avg. word tokens	21.7	16.2

3. Methodology

For both tasks, the same methodology was followed: first, two classical methods were implemented as a baseline and then, a series of BERT-based models were utilized to search for better results.

In the next subsection we present the Baseline models and the BERT-based models.

3.1. Baseline Models

Baseline models are both learnt on a TF-IDF matrix generated through the TfidfVectorizer class from the sklearn library. On each task, the vectorizer is fitted on the training dataset and then transformed on both the training and validation datasets. This training and validation TF-IDF matrixes are then used to train and validate the baseline methods. Before the TF-IDF matrixes are generated, the tweets follow a processing pipeline, which includes a Snowball Stemmer from the NLTK library.

- **Multinomial Naive Bayes**[3]. From the sklearn Python library, fitted on the input TF-IDF matrix and output labels with default parameters.
- **Linear SVC**[4]. From the sklearn Python library, fitted on the input TF-IDF matrix and output labels with a linear kernel and all other default parameters.

3.2. BERT-based Models

Transformer models are implemented by downloading the pre-trained base models from Hugging Face and then fine-tunning them on the text data using Tensorflow, Keras and the Transformers library.

The text data is not processed as before in the case of these models but with their own built-in tokenizers through the Transformers library. Additionally, in the case of RoBERTuito, the tweets need to be pre-processed with the pysentimiento library.

- **bert-base-spanish-wwm-cased (BETO).** BETO[5] is a Spanish version of the BERT-Base model.
- **robertuito-base-uncased.** RoBERTuito[6] is a RoBERTa implementation for social media text in Spanish trained on 500 million tweets.
- **mdeberta-v3-base.** mDeBERTaV3[7] is a multilingual implementation of the DeBERTa architecture.

4. Experimentation

4.1. Track 1

In this task the goal is to train and validate each of the selected models to a multi-class problem where the input data is the text data (tweets) from Track 1 and the output data are the assigned labels.

As explained in section 3, we have participated in this task with two baselines and three BERT-based Transformer models.

When working with the BERT-based models, the checkpoints are downloaded from their repository at Hugging Face and then loaded as Tokenizers that are used to process both training and validation tweets. The model is then instantiated using the TFAutoModelForSequenceClassification class from the Transformers library using the same checkpoint and the parameters and hyperparameters are tuned in order to get the best results. The TFAutoModelForSequenceClassification class is adapted to a multi-class problem. We have tuned the number of epochs,

batch size, start and end learning rates on the polynomial scheduler and the dropout probability as shown in Table 2.

Then the model is compiled and fitted with Tensorflow and Keras. In the compilation step, the AdamW[8] algorithm is always applied with a polynomial learning rate scheduler. After the model is fitted to the training data, it is validated to the validation dataset using weighted-average F1-Score, Precision and Recall and Accuracy.

Regarding performance, it is worth mentioning that all Transformer models are fine-tuned in Google Colab using NVIDIA Tesla T4 or V100 GPUs expect for the mDeBERTaV3 model, for which an NVIDIA A100 is utilized.

In the case of RoBERTuito, as we figured that this model displayed more tolerance than the others to tuning techniques, we added a class weight dictionary {NP: 0.7, NR: 1, P: 1.3} for the training phase to take care of the class imbalance situation that specially affects the model's ability to identify LGBT+phobic (P) tweets (minority class). This set of weights was manually tweaked.

Table 2
Experimentation set-up for Track 1

Model	Epochs	Batch size	start Learning rate	end Learning rate	dropout
BETO	4	4	$5e^{-5}$	0	-
RoBERTuito	7	4	$2e^{-5}$	0	0.2
mDeBERTaV3	5	32	$5e^{-5}$	0	0.2

As shown in table 3 weighted-average F1-Score, Precision and Recall and Accuracy were used to benchmark the models implemented in Track 1. RoBERTuito achieved the best results across all metrics, though neither this model nor any other Transformer-based model achieved a significant improvement over the simple Linear SVC baseline trained on TF-IDF matrixes, which doesn't take into account word order or semantic similarity between tokens as Transformer-based models do.

Table 3
Track 1 results obtained on validation data based on weighted F1-score, Precision and Recall and Accuracy

Model	w.F1-Score	w.Precision	w.Recall	Accuracy
Multinomial Naive Bayes	0.76	0.94	0.68	0.68
Linear SVC	0.84	0.85	0.83	0.83
BETO	0.86	0.87	0.86	0.86
RoBERTuito	**0.88**	**0.88**	**0.88**	**0.88**
mDeBERTaV3	0.83	0.83	0.84	0.84

4.2. Track 2

In this track the goal is to train and validate each of the selected models to a multi-label problem where the input data is the text data (tweets) from Track 2 and the output data are the multiple assigned labels.

Similar to Track 1, we have worked using the same procedure using Hugging Face models and the Tokenizer class. In the same way, we use TFAutoModelForSequenceClassification class using the parameters and hyperparameters showed in table 4 and adapt them to the multi-label problem. Then the model is compiled and fitted with Tensorflow and Keras. In the compilation step, the AdamW[8] algorithm is also applied with a polynomial learning rate scheduler. Regarding this track, when the model is fitted to the training data, it is validated to the validation dataset using macro-average F1-Score, Precision and Recall and Accuracy.

As for Track 1, we have participated in this task with two baselines and three BERT-based Transformer models. In the case of performance, we have also fine-tuned all Transformer models in Google Colab with the same specifications followed in Track 1.

Table 4
Experimentation set-up for Track 2

Model	Epochs	Batch size	start Learning rate	end Learning rate	dropout
BETO	4	4	$5e^{-5}$	0	0.2
RoBERTuito	7	4	$2e^{-5}$	0	-
mDeBERTaV3	5	32	$5e^{-5}$	0	-

As shown in table 5, macro-average F1-Score, Precision and Recall and Accuracy were utilized to benchmark the models implemented in Track 2. RoBERTuito achieved the best result on Accuracy and BETO achieved the best results on macro-average Precision and Recall. Again, none of the Transformer-based models achieved a significant improvement over the simple Linear SVC baseline.

Table 5
Track 2 results obtained on validation data based on macro-average F1-score, Precision and Recall and Accuracy

Model	macro F1-Score	macro Precision	macro Recall	Accuracy
Multinomial Naive Bayes	0.18	0.20	0.17	0.78
Linear SVC	0.49	0.44	0.59	0.87
BETO	0.52	0.51	0.59	0.86
RoBERTuito	**0.52**	**0.48**	**0.58**	**0.88**
mDeBERTaV3	0.19	0.20	0.19	0.78

4.3. Discussion on Results

We have seen how in both tasks a simple Linear SVC model almost achieved the same results as our best performer model RoBERTuito. We didn't submit Linear SVC to the HOMO-MEX competition so we can only conclude inside our experimental setup that simpler models such as a Linear SVC will perform as good as Transformer models if properly tuned because in the context of short social media texts, in this experimentation, language models don't seem to outperform in the classification problem. We can conclude then, that the existence (or lack of) LGBT+phobic terms and their type are the most important features in the context of both classification tasks, as the performance of such a simple statistical method over term frequencies demonstrates.

5. Conclusions

We have presented our participation in the shared task Homo-Mex 2023. We have participated in both tracks obtaining a second position according to official metrics reported by organizers. We have presented a set of experiments using classic algorithms and BERT-like Transformer models. The results obtained in both tasks exhibit similar performances for different methodologies.

6. Acknowledgements

This work was undertaken by the principal author in fulfilment of his Bachelor's Final Project at the Universitat Oberta de Catalunya.

References

[1] G. Bel-Enguix, H. Gómez-Adorno, G. Sierra, J. Vásquez, S.-T. Andersen, S.-L. Ojeda-Trueba, Overview of HOMO-MEX at Iberlef 2023: Paraphrase Detection in Spanish Shared Task, Procesamiento del Lenguaje Natural 71 (2023).

[2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186. URL: https://aclanthology.org/N19-1423. doi:10.18653/v1/N19-1423.

[3] C. Manning, P. Raghavan, H. Schuetze, Introduction to information retrieval, Cambridge University Press, 2008, pp. 234–265. URL: https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html.

[4] C. Manning, P. Raghavan, H. Schuetze, Introduction to information retrieval, Cambridge University Press, 2008, pp. 319–325. URL: https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html.

[5] J. Cañete, G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang, J. Pérez, Spanish pre-trained bert model and evaluation data, in: PML4DC at ICLR 2020, 2020.

[6] J. M. Pérez, D. A. Furman, L. Alonso Alemany, F. M. Luque, RoBERTuito: a pre-trained language model for social media text in Spanish, in: Proceedings of the Thirteenth Language Resources and Evaluation Conference, European Language Resources Association, Marseille, France, 2022, pp. 7235–7243. URL: https://aclanthology.org/2022.lrec-1.785.
[7] P. He, J. Gao, W. Chen, Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023. arXiv:2111.09543.
[8] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, 2019. arXiv:1711.05101.