eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Detección de discurso de odio en redes sociales

Autor

Pablo Felipe

2020

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Detección de discurso de odio en redes sociales

Autor

Pablo Felipe

Directores

Maite Oronoz, Rodrigo Agerri

Resumen

El hecho de que el discurso de odio supone un desafío para la sociedad es innegable. Éste es promovido a todos los niveles, desde por los líderes de todo el mundo hasta por la gente menos influyente, provocando en todos los escenarios casos de discriminación. Al tratarse de un problema tan generalizado, las redes sociales (RRSS) juegan un papel muy importante en el asunto debido a su uso en todo el mundo. Por lo tanto, la identificación de dichos comportamientos abusivos en estas plataformas puede ayudar en gran medida a analizar y lidiar con un desafío a nivel mundial como es la detección del discurso de odio.

En este Trabajo de Fin de Grado se analizan arquitecturas neuronales (desde clasificadores lineales hasta las últimas arquitecturas llamadas Transformers) para la detección de expresiones de odio en RRSS en distintos idiomas. En concreto, además de en castellano e inglés, se analiza en profundidad la aplicación de estos sistemas en euskera, donde la cantidad de corpus existente es muy limitada. Además, se recoge un corpus de tuits en euskera para el entrenamiento de modelos en este idioma.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

1.1. Definición de mensaje de odio

Una de las razones principales por las que la detección automática de discurso de odio supone un desafío es su misma definición. Al tratarse de un tema muy subjetivo, pueden surgir problemas a lo largo de la creación de sistemas para su detección: al realizar la búsqueda de corpus, anotar los *datasets* e incluso al evaluar los resultados del modelo entrenado. Por ello, consideramos oportuno mencionar algunas de las definiciones recogidas por [MacAvaney et al., 2019]:

- Facebook: "*Consideramos discurso de odio o lenguaje que incita al odio todo contenido que ataca directamente a las personas por lo que denominamos "características protegidas": raza, etnia, nacionalidad, religión, clase, orientación sexual, sexo, identidad sexual y discapacidad o enfermedad grave.*"
- Twitter: "*Conductas de incitación al odio: no se permite fomentar la violencia contra otras personas ni atacarlas o amenazarlas directamente por motivo de su raza, origen étnico, nacionalidad, pertenencia a una casta, orientación sexual, género, identidad de género, afiliación religiosa, edad, discapacidad o enfermedad grave. Tampoco permitimos la existencia de cuentas cuyo objetivo principal sea incitar la violencia contra otras personas en función de las categorías antes mencionadas.*"
- Encyclopedia of the American Constitution: "*Hate speech is speech that attacks*

> *a person or group on the basis of attributes such as race, religion, ethnic origin, national origin, sex, disability, sexual orientation, or gender identity.*"

Algunos ejemplos de discurso de odio en redes sociales

Como ya hemos visto, lo que tienen en común todas estas definiciones es el tipo de víctimas, ya que que los mensajes de odio se realizan por motivo de raza, origen étnico, nacionalidad, orientación sexual, género, identidad de género, religión, edad, discapacidad o enfermedad grave. No se incluyen los mensajes cuya connotación sea hiriente pero la razón del odio no sea que la víctima pertenezca a uno de esos colectivos. Para ilustrar las definiciones propuestas, a continuación se recogen con unos ejemplos reales extraidos del *dataset* HatEval en castellano e inglés (más detalles en 4.1.1).

Ejemplos de tuits que consideramos discurso de odio:

- *"tu vete a la verga, puto sudaca"*
- *"Qué es un partido doble mixto de tenis? Un single de hombres con obstáculos."*
- *"#YesAllWomen should stay in the kitchen"*
- *"Stop tweeting about football. You're a girl and you opinion doesn't count. #WomenSuck"*

Ejemplos de tuits que, a pesar de tener connotaciones agresivas o hirientes, no consideramos discuros de odio:

- *" eres una puta mierda"*
- *"shut the fuck up and come suck my dick"*

1.2. Motivación

Siendo el *Natural Language Processing* o Procesamiento del Lenguaje Natural (PLN) un campo en un inicio desconocido para mí, decidí realizar mi Trabajo Fin de Grado (TFG) sobre este tema por diversos motivos. Si el hecho de conseguir que un ordenador interprete texto natural por sí solo convierte al PLN en un campo interesante y con mucho potencial,

lo que verdaderamente me apasiona del mismo son sus aplicaciones. Al trabajar con un recurso tan común como es el texto, las aplicaciones del PLN pueden abarcar todo tipo de ámbitos sociales. Uno de los que me resulta más interesante (y que aborda este proyecto) es el de la detección de odio en redes sociales.

Vivimos en un tiempo donde la opinión de la gente es más pública y accesible que nunca. Es innegable que las redes sociales juegan un papel clave en el panorama político y social actual, ya que facilitan el intercambio opiniones e ideas entre personas de todo el mundo y dan lugar a movimientos de cualquier tipo y signo político. Por supuesto, el discurso de odio no se queda atrás.

Como vivimos en una sociedad multilingüe y en la actualidad no existen sistemas capaces de discernir el discurso de odio en euskera, consideramos que la creación de un modelo multilingüe capaz de detectar de manera fiable éstos comportamientos de odio puede ser de gran utilidad.

La segunda motivación es que este proyecto supone mi primer contacto con un ambiente de investigación real, y todo este aprendizaje me sería de ayuda en futuros proyectos profesionales y/o académicos. Además de la planificación y seguimiento que ello conlleva, el ritmo de aprendizaje en este tipo de proyectos es enorme debido a la necesidad de estar al día en temas tan populares como los que tratamos, donde a cada mes se publican técnicas y acercamientos basados en aprendizaje profundo (o *deep learning*) en el área del procesamiento del lenguaje.

1.3. Contribuciones

Éstas son las principales contribuciones de este Trabajo de Fin de Grado (TFG):

- Se proponen **sistemas para la detección de odio multilingüe** en el área del Procesamiento del Lenguaje Natural, con especial énfasis en la detección en textos en euskera. Se tratan de modelos entrenados mediante aprendizaje profundo cuyo corpus de entrenamiento está compuesto por tuits, el mensaje de texto estándar de Twitter.

- Se genera un *dataset* de tuits en euskera recogidos durante los meses de abril y mayo de 2020, que coinciden con la cuarentena por el COVID-19.

- Se genera un *dataset* de tuits en euskera con mensajes de odio (HatEval-EU), el primero disponible de este tipo.
- Se investiga la transferencia de conocimiento cross-lingüe para la detección de discurso de odio.
- Se investiga la relación entre la polaridad de un mensaje y su carga de odio.

1.4. Estructura del documento

El documento ha sido organizado en los siguientes capítulos:

- En el Capítulo 1 se realiza una **introducción** al tema del TFG, aportando definiciones, explicando las motivaciones para realizarlo y enumerando las contribuciones del mismo.
- En el Capítulo 2 se recogen los **objetivos principales del proyecto**.
- En el Capítulo 3 se realiza una visión general del **estado del arte**, revisando los *datasets* más utilizados para la creación de sistemas capaces de detectar odio, así como los sistemas y técnicas con mejores resultados en dicho campo del Procesamiento del Lenguaje Natural.
- En el Capítulo 4 se trata la **metodología del proyecto**. Se recogen los *datasets*, sistemas y aproximaciones posibles para el entrenamiento de sistemas capaces de detectar mensajes de odio. Además, se enumeran las herramientas utilizadas en el TFG para la recolección de *datasets*, el entrenamiento de modelos y la evaluación de los mismos.
- En el Capítulo 5 se describe la **experimentación** llevada a cabo a lo largo del proyecto para la creación de sistemas para la detección de mensajes de odio y medición de polaridad. Se recogen también los resultados de la evaluación de los modelos.
- En el Capítulo 6 se encuentran las **conclusiones** del proyecto junto con una breve reflexión de lo que ha supuesto a nivel académico y personal. Se reflexiona también sobre el trabajo futuro relacionados con este TFG.
- Finalmente, en el Anexo A se recoge el **Documento de Objetivos del Proyecto**.

2. CAPÍTULO

Objetivos del proyecto

El **objetivo principal** del proyecto es la aplicación de **modelos multilingües para la detección automática de mensajes de odio en redes sociales (RRSS)** mediante el Procesamiento del Lenguaje Natural (PLN o NLP por sus siglas en inglés). Para ello, primero debemos realizar un análisis exhaustivo de las técnicas y algoritmos más recientes y con los mejores resultados en el entrenamiento de modelos de lenguaje. Será imprescindible analizar aplicaciones recientes de técnicas de aprendizaje profundo en el área de la detección de discurso de odio. Con toda esta información obtendremos una visión general del panorama actual del PLN aplicado en la detección de odio en RRSS.

Otro objetivo del proyecto es conseguir un sistema capaz de detectar discurso de odio en euskera. Para ello, queremos entrenar un modelo de lenguaje mediante la aplicación de técnicas de aprendizaje profundo con el cual podamos predecir si un texto en euskera contiene odio o no. Otro de los objetivos principales es investigar la transferencia de conocimiento cross-lingüe para la detección de discurso de odio realizando experimentos con los modelos muiltilingües.

Por otro lado, se propone la creación de un *dataset* de tuits en euskera con el cual seamos capaces de entrenar sistemas para la detección de odio en dicho idioma. Para anotar este corpus "nativo", se plantea el uso de *Zero-shot learning*. Es decir, realizar predicciones en un corpus (el que vamos a crear) del cual el sistema no conoce ningún ejemplo.

Con el fin de complementar el análisis de los mensajes de odio, finalmente se medirá también la polaridad de los *datasets* con los que trabajamos. Como contienen un alto nivel de discurso de odio, métricas como la polaridad de las opiniones de los usuarios

pueden ser muy útiles para encontrar relaciones entre odio y negatividad en las opiniones y sacar conclusiones en base a los resultados.

3. CAPÍTULO

Estado del arte

En este apartado se realizará una visión general de los conjuntos de datos o *datasets* más utilizados para la creación de sistemas para la detección de odio, así como de las técnicas más utilizadas actualmente y con mejores resultados en dicho campo del Procesamiento del Lenguaje Natural.

3.1. Datasets

Actualmente existen muchos *datasets* para el entrenamiento de sistemas capaces de detectar mensajes de odio (la Tabla 3.1 recoge algunos de los más populares). Sin embargo, la cantidad de *datasets* existentes en unos idiomas es mucho mayor a la de otros. Por ejemplo, actualmente no existen *datasets* de mensajes de odio en euskera. Entre los recogidos en este capítulo, hay cinco en inglés y únicamente uno en castellano.

Uno de los puntos a tener en cuenta de estos corpus es la fuente de la que se ha extraído el texto, ya sean periódicos, libros, artículos, actas parlamentarias... Por ejemplo, en [de Gibert et al., 2018] construyen un *dataset* de mensajes extraídos del foro supremacista blanco *Stormfront*. Los mensajes se anotan con las siguientes etiquetas: *Hate* para los mensajes con discurso de odio, *NoHate* para los que no lo tienen o *Relation* para los que pueden tener esa connotación en ciertos contextos.

Sin embargo, la fuente de texto que buscamos en este proyecto son las RRSS. En concreto Twitter, pues se trata de una plataforma que es adecuada para la búsqueda, extracción y

Nombre	Lugar	Etiquetas	Tuplas	Idioma	Publicación
Waseem and Hovy	Twitter	Racism-sexism-none	16.914	Inglés	2016
Waseem	Twitter	Racism-sexism-none-both	20.947	Inglés	2016
Stormfront	Stormfront	Hate-noHate-Relation	10.568	Inglés	2017
Hatebase	Twitter	Hate-Offense-None	24.802	Inglés	2017
Hateval-EN	Twitter	Hate/noHate	13.000	Inglés	2019
Hateval-ES	Twitter	Hate/noHate	6.600	Español	2019

Tabla 3.1: Datasets en el estado del arte sobre discurso de odio.

análisis de mensajes de odio. Uno de los *datasets* formados por tuits más populares es el de [Davidson et al., 2017]. El corpus es obtenido mediante la búsqueda de tuits que contuvieran una serie de palabras clave (con connotación de odio). Estas palabras son las recogidas por *Hatebase*[1], una plataforma donde se agrupan más de 3.500 términos con connotaciones de odio en 97 idiomas distintos. Además de *Hatebase*, de entre los *datasets* formados por tuits destacan [Waseem and Hovy, 2016] y [Waseem, 2016], ambos enfocados en racismo, sexismo y LGBT-fobia, entre otros.

El *dataset* más relevante para nuestro proyecto es el de HatEval [Basile et al., 2019]. Se trata de un *dataset* recopilado para la tarea 5 de la competición de SemEval 2019[2]. Está anotado con 3 etiquetas binarias para identificar si un tuit contiene odio, es agresivo y/o la víctima es una o un grupo de personas.

Ejemplos de los datasets

Para hacernos una idea del tipo de mensajes que se recogen en los *datasets* mencionados, es interesante recoger unos ejemplos.

Dataset Stormfront

Cada tupla del *dataset* contiene los IDs del mensaje, el usuario y el subforo, así como un indicador sobre la relevancia del contexto y la etiqueta del mensaje (ver Tabla 3.2).

Para poder visualizar el mensaje (todos en inglés), es necesario acceder a unos ficheros auxiliares donde se almacenan todos los mensajes con un id (el del *dataset*). Buscando la correspondencia, nos encontramos con los siguientes mensajes:

[1] https://hatebase.org/
[2] http://alt.qcri.org/semeval2019/

file_id	user_id	subforum_id	num_contexts	label
12834217_3	572066	1346	0	noHate
12860820_1	573375	1346	0	hate

Tabla 3.2: Ejemplos de Stormfront

1. *Simply copy and paste the following text into your YouTube videos description boxes (no hate)*
2. *Greece need a man like Adolf Hitler and now is the perfect time to start a national political movement to free Your country from the zionist bankers (hate)*

Podemos ver que en un mismo subforo se encuentran mensajes de todo tipo, desde indicaciones para utilizar una página web hasta mensajes claramente politizados. El modelo entrenado con este corpus aprenderá con todos ellos.

Dataset Davidson

La anotación de este *dataset* en inglés se realizó mediante CrowdFlower (plataforma para el etiquetado colaborativo de *datasets*) y cada tuit fue anotado por varios usuarios de dicha plataforma. Para ilustrarlo (ver Tabla 3.3), cada tupla del *dataset* contiene (además del propio tuit, el índice y la etiqueta final) el número de personas que etiquetaron el tuit (etiqueta *All*), cuántos lo etiquetaron como *hate* u odio (etiqueta *HS*), cuántos como *offensive* u ofensivo (etiqueta *Off*) y cuántos como *neither* o ninguno (etiqueta *None*). Finalmente, la etiqueta *class* indica cuál es la etiqueta final (0 para HS, 1 para Offensive y 2 para None).

Datasets Hateval

La anotación de los *datasets* de HatEval también se realizó mediante una plataforma de anotación en masa, solo que esta vez el número de anotadores no se refleja en el *dataset* (ver Tabla 3.4). Los mensajes de odio en el *dataset* se dan en todo tipo de situaciones, desde conversaciones entre usuarios de la red social hasta en discusiones sobre series de televisión. Cada tupla del *dataset* tiene la siguiente estructura: un *id* para identificar la tupla en cuestión, una columna *text* con el texto del tuit y tres etiquetas binarias para identificar si tiene discurso de odio (etiqueta *HS*), si la victima es una persona o un grupo (etiqueta *TR*) y si el tuit contiene agresividad (etiqueta *AG*). Existen dos *datasets*: uno con tuits en castellano (HatEval-ES) y otro con tuits en inglés (HatEval-EN).

All	HS	Off.	None	Class	Tweet
3	3	0	0	0	We hate niggers,we hate faggots and we hate spics-kkk rally
3	0	3	0	1	All these hoes been passed around to a Cleveland niggah
3	0	1	2	2	"momma said no pussy cats inside my doghouse"

Tabla 3.3: Ejemplos de Davidson.

id	text	HS	TR	AG
101	@*** @*** @*** Oye por qué no molestas a tu puta madre?	0	0	0
166	Los árabes son los hombres más descarados y ligadores del mundo	1	0	1
138	Te odio Lidia, no te mereces a Francisco, #perra #LasChicasDelCable2	1	1	1

Tabla 3.4: Ejemplos de Hateval-ES.

En ocasiones, los ejemplos de tuits mostrados en el proyecto contendrán menciones de usuarios de Twitter (@usuario). Éstos serán ocultados, sustituyendo el nombre de usuario por asteriscos (@***).

3.2. Sistemas

En este apartado se describirán sistemas con resultados estado del arte en detección automática de odio. Nos centraremos en los sistemas creados a partir de *datasets* formados por tuits, ya que son los sistemas en los que basaremos los nuestros. Los sistemas presentados en el desafío HatEval (apartado 4.1.1) son especialmente interesantes por el uso de diversidad de sistemas en distintos idiomas.

En la tarea de evaluación HatEval propuesta por SemEval en 2019 se presentaron un total de 108 sistemas: 69 en inglés y 39 en castellano.

Las técnicas utilizadas para la creación de estos sistemas abarcan desde técnicas de aprendizaje automático (que han resultado especialmente prometedoras en el ámbito del procesamiento de texto, como podrían ser las Redes Neuronales Recurrentes (RNR), redes neuronales tipo *Long Short-Term Memory* (LSTM) [Sherstinsky, 2018], *bidirectional LSTM* (BiLSTM)... más en el apartado 4.2.2) hasta sistemas centrados en el análisis de características lingüísticas (siendo estos los menos comunes de los sistemas propuestos).

Sistemas presentados para el tarea de evaluación en inglés de HatEval

El equipo Fermi [Indurthi et al., 2019] desarrolló el mejor sistema en inglés con un F1-score de 0,6510. El sistema consistía en un modelo de máquinas de vectores de soporte o *Support Vector Machine* (SVM) entrenado con kernel Radial Basis Function (RBF), utilizando a su vez embeddings de Google's Universal Sentence Encoder [Cer et al., 2018]. En puestos inferiores se encuentran, entre otros, sistemas basados en Redes Neuronales Convolucionales (RNC) y *Long Short Term Memory* (LSTM).

Sistemas presentados para la tarea de evaluación en castellano de HatEval

Por otro lado, en la tarea de evaluación en castellano los equipos con mejores resultados llevan a cabo un acercamiento similar al equipo Fermi. Tanto el equipo Atalaya [Pérez and Luque, 2019] como MineriaUNAM empataron con un F1-score de 0,7300 mediante el uso de modelos SVM. Atalaya entrenó su sistema con representaciones de texto obtenidas mediante *sentiment-oriented word vectors* de fastText [Bojanowski et al., 2016]. MineriaUNAM, sin embargo, optó por buscar la mejor combinación entre características de patrones lingüísticos, diccionarios de palabras agresivas y distintos tipos de N-gramas (estructuras de N palabras consecutivas).

Comparación de resultados y conclusión

Podemos observar que el valor F1 máximo obtenido en la tarea de evaluación en castellano (empatan los equipos Atalaya y MineriaUNAM con un 0,73) es mayor que el obtenido en inglés (0,6510 por el equipo Fermi). La diferencia se hace más notable con las medias de cada tarea de evaluación, siendo en inglés de un 0,4484 y en castellano de un 0,6821 (Tabla 3.5).

Tarea de evaluación en castellano	
Models	F1-score
Atalaya Team	**0,7300**
MineriaUNAM Team	**0,7300**
Media de la competición	0,6821
Tarea de evaluación en inglés	
Models	F1-score
Fermi Team	**0,6510**
Media de la competición	0,4483

Tabla 3.5: Comparación entre los sistemas presentados a HatEval

Tanto en la tarea en castellano como en la de inglés los algoritmos SVM (de la familia de los clasificadores lineales, más información en 4.2.1) obtienen los mejores resultados entre los sistemas presentados. Sin embargo, no vemos algoritmos con arquitecturas neuronales entre los sistemas con mejores evaluaciones porque las experimentaciones con arquitecturas como la Transformer (más información en 4.2.2) son posteriores a la tarea HatEval. Como los sistemas basados en la arquitectura de Transformers están obteniendo resultados muy competitivos para un gran número de tareas de PLN, encontramos una motivación en probar esta arquitectura para la creación de sistemas capaces de detectar odio en redes sociales, ya que podríamos comparar su rendimiento con los demás sistemas presentados en la competición HatEval de 2019 y ver si realmente los Transformers están a la altura de los sistemas más populares.

En el desafío 12 de SemEval 2020 [Zampieri et al., 2020] (distinto a HatEval) se discute de nuevo sobre sistemas capaces de detectar odio en Internet. En este nuevo desafío sí se utilizan arquitecturas neuronales para la creación de sistemas. De hecho, la mayoría de sistemas participantes parten de modelos pre-entrenados como XLM-RoBERTa [Liu et al., 2019], justamente el modelo que utilizamos nosotros para nuestros sistemas Transformers. Sin embargo, los resultados del desafío fueron publicados el 12 de junio de 2020, cuando el desarrollo de éste Trabajo de Fin de Grado ya estaba en fase avanzada de desarrollo. Por tanto, los resultados obtenidos en éste TFG serán comparados y evaluados junto a los sistemas de HatEval (2019).

4. CAPÍTULO

Metodología

Previamente se ha realizado una visión general de los sistemas más populares en el ámbito de la detección de odio mediante el Procesamiento de Lenguaje Natural (PLN o NLP por sus siglas en inglés). En este apartado se realizará una visión general tanto de los *datasets* utilizados a lo largo del proyecto como de las distintas aproximaciones posibles para el entrenamiento de sistemas capaces de detectar mensajes de odio. Además, se enumeran todas las herramientas utilizadas en el proyecto para la recolección de *datasets* y entrenamiento de modelos. Finalmente, se explicarán las métricas utilizadas para la evaluación de los sistemas desarrollados en el proyecto.

4.1. Datasets utilizados

En el proyecto se han utilizado *datasets* recogidos por terceros y creados por nosotros para la tarea de detección de odio.

4.1.1. HatEval

Se trata del *dataset* utilizado para la competición HatEval sobre sistemas capaces de detectar discurso de odio que organizó SemEval en 2019. Además, es uno de los *datasets* más relevantes de este Trabajo de Fin de Grado.

El corpus del *dataset* original está formado por un total 19600 tuits, siendo 13000 en

inglés y 6600 en castellano (el corpus está dividido en dos *datasets* según el idioma, HatEval-ES y HatEval-EN). La distribución en cuanto a las víctimas de odio de los tuits son de 9091 sobre inmigrantes y 10509 sobre mujeres. Cada uno de los tuits está anotado con 3 clases binarias:

- Discurso de odio (HS) : El tuit contiene odio (1) o no (0)
- Target individual (TR): La víctima es un solo individuo (1) o un grupo de personas (0)
- Agresividad (AG): El tuit contiene agresividad (1) o no (0)

En ambos idiomas el corpus está dividido en las tres partes que de manera estándar se utilizan para el entrenamiento de modelos mediante aprendizaje automático: las particiones de desarrollo (o *dev*) y entrenamiento (o *train*) se utilizan para el entrenamiento del modelo y la partición de prueba (o *test*) se utiliza para la evaluación del mismo. Este corpus se ha recogido de la página web de la tarea[1].

Problemas con HatEval-EN

El *dataset* en inglés contiene graves errores de anotación. Esta cuestión se analiza en el artículo [Stappen et al., 2020], donde recogen 2.971 ejemplos del *dataset* en inglés y descubren que 1.564 de ellos presentan falsos positivos. Es decir, de entre el subconjunto de ejemplos, más del 50 % de los casos son etiquetados positivos erróneamente.

Según el artículo, el problema se podría deber a que la partición de prueba no fue recogida junto a las particiones de entrenamiento y desarrollo. Esa diferencia de métodos y criterios de búsqueda aplicados para recopilar la partición de prueba respecto a los utilizados para recoger las demás, ya sea por temática o por la diferencia temporal, hace que la distribución de datos en el dataset completo no estuviera balanceada.

Como consecuencia, al entrenar modelos con las particiones de desarrollo y prueba (recogido para HatEval, centrado en odio contra inmigrantes) y después evaluarlo con el subset de prueba (*dataset* Fersini, centrado en odio contra mujeres) con características muy distintas, dio lugar a una gran cantidad de falsos positivos en las predicciones.

[1] https://competitions.codalab.org/competitions/19935

	Tuit
Original	@KurtSchlichter LEGAL is. Not illegal. #BuildThatWall
Preprocesado	LEGAL is. Not illegal.

Tabla 4.1: Antes y después de preprocesar.

Adaptación del dataset

Todas las versiones del *dataset* HatEval utilizados en nuestro proyecto contienen una única etiqueta binaria: Hate / No-Hate. Se toma esa decisión con el fin de centrar todos los esfuerzos en conseguir un sistema capaz de detectar odio, dejando atrás información complementaria como la agresividad o el número de víctimas de los mensajes.

Uno de los inconvenientes de que el corpus de HatEval original esté formado por tuits es que está repleto de elementos que pueden estorbar en el entrenamiento de modelos. Entre éstos se pueden encontrar los *Hashtags* ('#', carácter utilizado para las tendencias, muy comunes en Twitter), las menciones a usuarios (@), las URLs, etc. Para deshacernos de ellos, utilizamos la librería *tweet-processor* [2].

De esta manera, sumamos las versiones preprocesadas de los *datasets* HatEval-ES y HatEval-EN a la lista de *corpora* que se utilizan en el proyecto.

Consecuencias del preprocesado. Pese a eliminar mucho texto que puede ser irrelevante para entrenar nuestros modelos, al realizar el preprocesado es posible que eliminemos información que puede ser útil. En el ejemplo de la Tabla 4.1, podemos ver un tuit etiquetado como odio en su versión original y preprocesado. En el original, existen dos elementos que son eliminados posteriormente: una mención a un abogado y escritor estadounidense y una tendencia con connotaciones racistas. Por un lado, al eliminarlos evitamos entrenar un modelo que aprenda a identificar odio en base a unas tendencias concretas de Twitter. Esto es importante, pues es posible existan mensajes con esa tendencia pero que a la vez no contengan mensaje de odio. Además, el modelo podría ser utilizado en contextos distintos a Twitter porque el texto utilizado para entrenarlo no contiene elementos específicos de la red social. Por otro lado, un modelo entrenado con las tendencias podría llegar a identificar odio en tuits de manera más eficaz (teniendo en cuenta tendencias con connotaciones racistas), pero sería menos polivalente y su uso en contextos ajenos a Twitter sería más limitado.

[2] https://github.com/s/preprocessor

HatEval EN + ES

Por otro lado, se genera un nuevo *dataset* Hateval mediante la unión de dos distintos: el de inglés y el de castellano. El fin de la fusión de estos *datasets* es entrenar modelos en ambos idiomas a la vez. Esta unión se realizó juntando los cada *subset* de un *dataset* con su *subset* correspondiente en el otro. Es decir, se juntó la partición de entrenamiento de HatEval-ES con la partición de entrenamiento de HatEval-EN, etc.

HatEval-EU

Uno de los objetivos de nuestro proyecto es el de crear un sistema capaz de detectar discurso de odio de manera similar a HatEval, solo que en nuestro caso lo haremos en euskera. La escasez de corpus en este idioma supone un gran problema a la hora de crear modelos competentes mediante aprendizaje automático.

La idea de generar este *dataset* surge como alternativa a la creación de un *dataset* nativo con tuits en euskera en caso de que este último no contuviera la suficiente carga de odio como para entrenar con él los sistemas para la detección de discurso de odio que buscamos.

Para crear un dataset en euskera alternativo al nativo, la traducción al euskera de *datasets* en otros idiomas fue una de las opciones más tentadoras. La disponibilidad de corpus sobre discurso de odio en idiomas como el inglés o el castellano es mayor a la del euskera y una traducción de los mismos podría dar lugar a *datasets* con la suficiente carga de odio como para entrenar sistemas para detectar discurso de odio en euskera con ellos.

Ésto se lleva a cabo traduciendo cada uno de los tuits del *dataset* HatEval preprocesado manteniendo las etiquetas. Como ya hemos visto, el dataset HatEval-EN contiene graves errores de anotación. Por tanto, evitamos utilizar este *dataset* y en su lugar traducimos al euskera HatEval-ES, obteniendo así nuestro dataset traducido HatEval-EU.

El sistema elegido para traducir el *dataset* es Opus-MT [Tiedemann and Thottingal, 2020]. En concreto, utilizaremos el modelo ‘opus-mt-es-eu’ para la traducción de castellano a euskera.

Algunos ejemplos de HatEval-EU

A continuación se muestran tuits del corpus HatEval-EU comparados con los originales de HatEval-ES. Consideramos que, en general, la traducción mediante el sistema Opus-

HatEval-ES	HatEval-EU	Etiqueta
Lastima.... callate zorra!!!!!!!	Min ematen dit.... Ixo puta!!!!!!!!	Odio
Callate puta	Ixo puta!	Odio

Tabla 4.2: Antes y después de traducir.

MT ha sido correcta. En la Tabla 4.2 se recogen algunos ejemplos de esta traducción. Se tratan de traducciones literales donde el sentido de algunas expresiones puede perder el sentido original. Si esta pérdida es demasiado grande, es posible que el significado del tuit y su etiqueta cambien completamente con la traducción (apartado 5.3.2).

4.1.2. Dataset de tuits en euskera

La creación de un corpus mediante la recopilación de tuits en euskera se presentó como una opción viable debido al fácil acceso de los mismos mediante la *Application Programming Interface* (API) gratuita de Twitter. Una de las ventajas sería que los datos obtenidos serían muy recientes y podrían tratar temas relevantes del momento (y propensos al lenguaje de odio) como pueden ser el coronavirus y la gestión del gobierno, el odio entre vecinos en tiempos de pandemia, la tragedia de Zaldibar, etc. Como contrapunto, la creación de un *dataset* en euskera desde 0 tenía el riesgo de no contener las características necesarias para el entrenamiento de modelos robustos. Es decir, en caso de que la comunidad de twitter en euskera no mostrara las suficientes expresiones de odio, la creación de un sistema así sería inviable.

Como primera opción y sin descartar las demás, finalmente se propuso una recolección extensa de tuits en euskera para la creación de un corpus desde 0.

El proceso de creación del *dataset* de tuits en euskera se explica más detalladamente en el apartado 5.3.

4.1.3. Dataset Behagune: polaridad

Con el fin de complementar las predicciones de discurso de odio, en este proyecto se realizan también predicciones sobre polaridad. El *dataset* utilizado para entrenar el modelo capaz de hacerlo es el recopilado en [Agerri et al., 2020]. Este corpus está formado por tokens recogidos tanto de Wikipedia en euskera como de artículos y noticias de periódicos.

El corpus se divide en dos conjuntos de datos, uno en castellano y otro en euskera.

4.2. Aproximaciones

En este apartado se enumeran las distintas aproximaciones posibles para el entrenamiento de sistemas capaces de detectar mensajes de odio. En [Zhang and Luo, 2018] dividen los sistemas capaces de detectar odio en dos grupos: los métodos más clásicos basados en representaciones de características extraídas manualmente (utilizadas por clasificaciones lineales) y los más modernos basados en aprendizaje automático mediante el uso de redes neuronales.

4.2.1. Métodos clásicos

Dentro del primer grupo se encuentran técnicas que extraen manualmente las características de las palabras y las agrupan o codifican en vectores de características que después utilizarán los clasificadores. Un ejemplo de este tipo de representación de características pueden ser los Bag of Words (BoW) o bolsas de palabras, los cuales contienen diccionarios con las palabras de los textos con los que se quieren entrenar los modelos. En estos diccionarios se representa la relevancia de cada elemento mediante métricas como, por ejemplo, la cantidad de veces que un elemento se repite en los datos de entrada. La figura que vemos a continuación es un ejemplo simple de una BoW, donde se cuenta la ocurrencia de cada palabra de la primera fila en cada frase (*Review 1, 2 y 3*).

- *Review 1: This movie is very scary and long*
- *Review 2: This movie is not scary and is slow*
- *Review 3: This movie is spooky and good*

Se trata de un ejemplo muy simple donde la mayoría de palabras aparecen una vez o ninguna, a excepción del verbo *is*, que aparece dos veces en la *Review 2*

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

Figura 4.1: Ejemplo de una *bolsa de palabras*.
analyticsvidhya.com

El problema con estos sistemas es que dependen del uso de un conjunto de palabras muy concretas. Estas palabras pueden adoptar distintos significados dependiendo del contexto debido a elementos más complejos como el sarcasmo o la burla. En [MacAvaney et al., 2019] ponen como ejemplo de éste problema expresiones como *"build that wall"*, cuyo significado literal dista mucho de la connotación que pueda adquirir en el contexto político estadounidense.

Clasificadores lineales

Antes de pasar a las arquitecturas neuronales, es importante mencionar algunos de los algoritmos más populares actualmente en PLN: los clasificadores lineales. El funcionamiento de estos algoritmos consiste en clasificar en base a un valor obtenido de una combinación lineal de sus características. Algunos ejemplos de estos algoritmos son Naïve Bayes, la Regresión Logística y las máquinas de soporte vectorial o *Support Vector Machines* (SVM). Éste último es especialmente relevante, pues es muy popular en tareas de detección automática de discurso de odio como HatEval, donde los sistemas creados mediante SVM obtienen los mejores resultados de la competición (más información en 3.2).

4.2.2. Aprendizaje profundo

Por otro lado se encuentran los métodos basados en aprendizaje profundo o *deep learning*. Estos métodos, al contrario que los clásicos, extraen representaciones abstractas de los textos con los que se entrenan y utilizan redes neuronales (RN) para entrenar los modelos.

Redes Neuronales Recurrentes

Una de las arquitecturas de redes neuronales más populares son las Redes Neuronales Recurrentes (RNR). En este tipo de arquitecturas, la red neuronal se ejecuta iterativamente, actualizando los pesos y generando una salida a cada paso. Cada una de estas iteraciones recibe dos tipos de entradas: la salida del paso anterior y la representación vectorial de todos los pasos recorridos hasta el momento. Es decir, además de generar una salida, a cada paso la RN guarda la información del entrenamiento realizado y ésta pasa a formar parte de una representación vectorial. A su vez, esta representación será la entrada del siguiente paso de la RN. Al contrario que en una RN común, la salida generada por la RNR en cada paso no se ve únicamente influenciada por los pesos que pueda contener la red, sino que también se utiliza la representación de todos los pasos anteriores para determinar la salida.

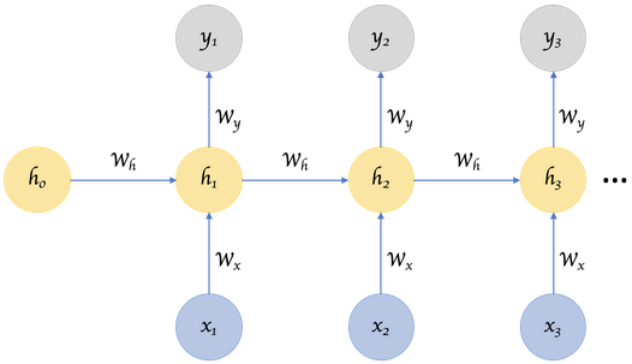

Figura 4.2: Esquema la arquitectura de *Redes Neuronales Recurrentes*. towardsdatascience.com

Sin embargo, esta arquitectura tiene limitaciones. Con un comportamiento ideal, la RNR sería capaz de recordar las palabras más importantes desde los primeros pasos hasta el último. En la práctica, la representación de la información almacenada en los primeros pasos tiende a desaparecer según avanza la ejecución. Esto se debe a que la RN se limita a representar las entradas de cada paso sin distinguir la información relevante de la que no lo es.

Long Short Term Memory

Con los modelos Long Short Term Memory (LSTM) se propone una forma de aliviar este problema. Se trata de una arquitectura basada en las RNR solo que, en este caso, se

establecen unos criterios para almacenar la información obtenida hasta el momento. El modelo aprenderá qué partes de la representación se deben olvidar para incluir las más importantes. En cada iteración se evaluará la entrada mediante unos criterios que actúan como puertas. Estas puertas dejarán pasar aquella información que cumpla los criterios (y pasará a formar parte de la representación de todos los pasos recorridos) y negará el paso a la que no los cumpla.

Por otro lado, existe una versión alternativa llamada BiLSTM. Se trata de una arquitectura idéntica a los LSTM, solo que en este caso la red neuronal se entrenará con los mismos datos una segunda vez recorriéndolos en orden inverso.

Si bien los LSTM/BiLSTM suponen una mejora respecto a las RNR clásicas, ambos modelos comparten una arquitectura secuencial que limita en gran medida la paralelización de las ejecuciones y, por tanto, el rendimiento general.

Transformers y el mecanismo de atención

En 2017 se propuso una nueva arquitectura para solventar esos problemas de rendimiento: los *Transformer* [Vaswani et al., 2017]. Con los LSTM se pretendía aprender qué parte de la representación recogida durante el entrenamiento se debía olvidar para introducir la nueva. Los Transformer, sin embargo, utilizan el mecanismo de atención. Como su nombre indica, consiste en aprender a mantener la *atención* sobre los elementos importantes.

Attention is all you need

Mediante el uso de este mecanismo se pretende dejar a un lado la idea de memorizar todos los datos recibidos hasta el momento. En su lugar, el modelo aprende a escoger los elementos según su relevancia.

Antes de pasar al mecanismo de atención, el Transformer necesita representaciones vectoriales de cada palabra. Además, estos vectores guardarán información sobre la posición de la palabra en la frase, dándole así un valor contextual (*positional encoding*). Para obtener valores contextuales generalmente se utilizan los llamados *embedding space*, que no es más que un espacio formado por representaciones vectoriales de palabras agrupadas según el contexto. Un algoritmo *state-of-the-art* para obtener representaciones vectoriales es GloVe [Pennington et al., 2014].

Una vez obtenidas las representaciones vectoriales, llega la clave de esta arquitectura: el *Multi-Head Attention*. Éste módulo calcula vectores de atención para cada una de las

palabras. En estos vectores se calcula la relevancia de cada palabra respecto a las demás. Por tanto, se calculan múltiples valores de atención en lugar de una sola suma ponderada de los valores como ocurre en otras arquitecturas.

Los Transformers utilizan dos módulos de atención: el codificador utiliza el bloque Multi-Head para obtener los vectores de atención y, más adelante, el decodificador utilizará esos vectores junto a otro bloque Multi-Head para obtener un resultado.

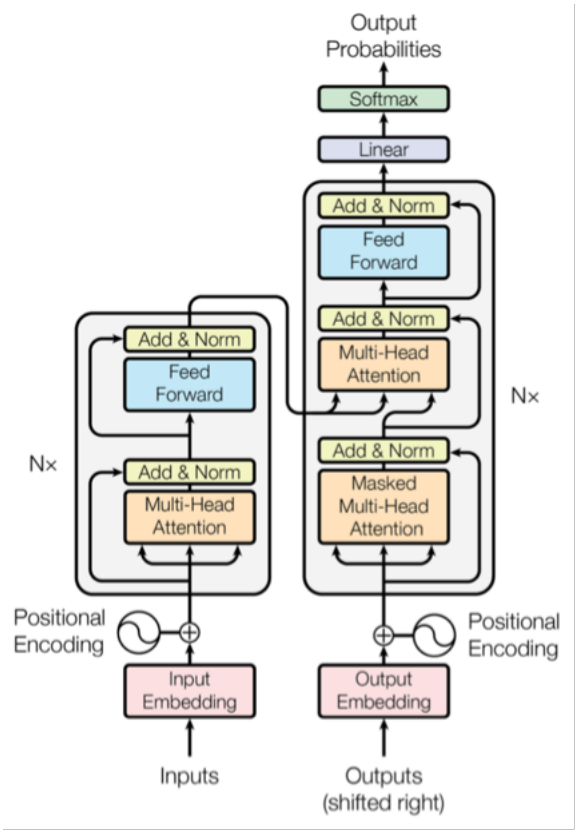

Figura 4.3: Arquitectura *Transformer*. A la izquierda se encuentra el codificador y a la derecha el decodificador. Ambos trabajan con módulos de atención.

4.3. Herramientas utilizadas

Hasta ahora se han explicado las aproximaciones más importantes dentro del área de la detección de odio mediante aprendizaje automático. En este apartado se hace una visión general de las herramientas utilizadas en este TFG.

En cuanto a los **sistemas**, en nuestros experimentos hemos usado:

- La librería fastText para el entrenamiento de modelos *baseline*, es decir, para poder establecer unos valores o límites inferiores a los resultados que se obtienen más adelante con modelos más complejos.

- Mediante la librería HunggingFace's Transformers [Wolf et al., 2019], utilizamos la arquitectura de *Transformers* para entrenar modelos contextuales multilingües para la detección de discurso de odio en el conjunto de textos en castellano (HatEval-ES), inglés (HatEval-EN), la fusión entre ambos y euskera (HatEval-EU). En lugar de entrenar estos modelos desde 0, partimos del modelo pre-entrenado XML-RoBERTa, disponible también en la librería de HuggingFace. Con este modelo ya entrenado, realizaremos un segundo entrenamiento llamado *Fine-Tune* (apartado 4.3.2), con el cual podremos especializar el modelo en una tarea más concreta con un nuevo *dataset*. Este entrenamiento se realizará con el código del programa de prueba GLUE (apartado 4.3.2), también incluido en la librería HuggingFace's Transformers.

Además, en este apartado mencionamos también las herramientas utilizadas para construir el **dataset** nativo en euskera. La *Application Programming Interface* (API) facilitada por Twitter es la piedra angular de la recogida del corpus, ya que gracias a ella somos capaces de buscar, filtrar y recoger tuits de manera sencilla.

4.3.1. fastText

Con el objetivo de establecer un límite inferior de lo que esperamos de nuestros futuros modelos, entrenamos dos modelos simples en inglés y castellano. Para ello, utilizamos la librería fasText [Joulin et al., 2016].

fastText es una librería desarrollada por el equipo de investigación de Inteligencia Artificial (IA) de Facebook para el aprendizaje de *word embeddings* o vectores de palabras y modelos supervisados y no-supervisados. Nosotros lo usaremos para obtener modelos de clasificación de texto (etiquetar odio/no-odio).

Para entrenar los dos modelos, se utilizarán los *datasets* ofrecidos por SemEval 2019 para la competición HatEval. fastText utiliza clasificadores lineales para entrenar los modelos mediante aprendizaje supervisado.

Formato para el corpus de entrenamiento

Para poder entrenar los modelos con los *datasets* de HatEval, tenemos que adaptarlos al formato de fastText: una primera columna con la etiqueta y, separado por un tabulador, una segunda columna con el texto del tuit al que corresponde la etiqueta (ver Tabla 4.3).

[__label__] [tab] [tuit]

Etiqueta	Texto
__label__0	@*** Tu eres un hijo de la gran puta
__label__1	@*** go rape your whore of a mother

Tabla 4.3: Ejemplos del formato fastText

Esta tarea se realiza mediante la Hoja de Cálculo de Google[3].

Utilidades de fastText

Para poder utilizar la librería, la descargamos del repositorio que ofrecen gratuitamente los propios desarrolladores[4].

Los comandos disponibles en fasText son:

supervised: entrenar un clasificador supervisado

quantize: cuantificar un modelo para reducir el uso de memoria

test: evaluar un clasificador supervisado

test-label: printear etiquetas con valores de precisión y cobertura (más en 4.4)

predict: predict most likely labels

predict-prob: predecir las etiquetas más probables y las probabilidades

skipgram: entrenar un modelo *skipgram*

cbow: entrenar un modelo *cbow*

print-word-vectors: dado un modelo entrenado, obtener vectores de palabras

[3]https://www.google.es/intl/es/sheets/about/
[4]https://github.com/facebookresearch/fastText.git

print-sentence-vectors: dado un modelo entrenado, obtener vectores de frases

print-ngrams: dado un modelo entrenado y una palabra, obtener ngramas

nn: buscar vecinos más cercanos

analogies: buscar analogías

dump: dump arguments,diccionarios, vectores entrada/salida

4.3.2. HuggingFace's Transformers y XLM-RoBERTa

Actualmente, una de las maneras más fáciles de utilizar modelos Transformers es mediante la librería de Transformers de HuggingFace [Wolf et al., 2019]. Esta librería tiene disponible una gran cantidad de modelos de distintas arquitecturas de entre los cuales destacamos XLM-RoBERTa [Liu et al., 2019]. Se trata de un modelo pre-entrenado en más de 100 idiomas que obtiene resultados *state-of-the-art* en el ámbito de la clasificación de texto (la detección de discurso de odio se encuentra en esta categoría). Ésta librería ofrece además *pipelines* con los cuales podremos utilizar los modelos ya mencionados para realizar predicciones. Por ejemplo, dados un modelo entrenado en detección de odio y un texto cualquiera, los pipelines pueden generar una salida en forma de etiqueta (en nuestro caso, *Odio* o *No odio*).

Todos los modelos (a excepción de los de fastText) entrenados en este proyecto serán versiones *Fine-Tuned* de XLM-RoBERTa utilizando el código de entrenamiento de sistemas de GLUE (apartado 4.3.2). A continuación explicaremos el concepto de *Fine-Tuning*.

Modelos multilingües, Fine-Tune y Zero-Shot

En el procesamiento de lenguaje natural es común el uso de modelos de lenguaje. La ventaja de un modelo multilingüe respecto uno estándar es que se pueden realizar predicciones sobre cualquier *dataset* cuyo idioma sea uno en los que el modelo fue entrenado. Además, cabe la posibilidad de volver a entrenar (tunear) ese modelo en un tema concreto (con un nuevo corpus). Así, aunque ya haya sido entrenado previamente, se especializa en el nuevo corpus para generar un nuevo modelo que pueda realizar predicciones sobre el tema en el que ha sido tuneado, sin perder el entrenamiento que recibió el modelo multilingüe pre-entrenado original. A este segundo entrenamiento se le llama Fine-Tune.

Debido a la versatilidad de estos modelos, nuestro objetivo es escoger un modelo pre-entrenado multilingüe (euskera incluido) y *Fine-Tunearlo* con el corpus de HatEval. Así, conseguiremos un modelo multilingüe especializado en discurso de odio que puede realizar predicciones sobre nuestro *dataset* de tuits en euskera sin haber recibido previamente ningún ejemplo del mismo. A este proceso de predicción "a ciegas" se le llama Zero-Shot.

El benchmark GLUE y la adaptación del código para *dataset* personalizados

El benchmark General Language Understanding Evaluation (GLUE) [Wang et al., 2018] se trata de una colección de herramientas para el entrenamiento, evaluación y análisis de sistemas PLN. Además de incluir una serie de *datasets* destinados a tareas distintas (entre los que se incluye el análisis de sentimientos), GLUE incluye código para poder crear sistemas con dichos *datasets*. Por suerte, la librería HuggingFace's Transformers permite utilizar fácilmente GLUE para entrenar modelos.

Sin embargo, el benchmark permite únicamente entrenar modelos con la colección de *datasets* que tiene incluida por defecto. Por tanto, es necesario adaptar el código del benchmark para entrenar modelos con un *dataset* personalizado. Es necesario también cambiar los *datasets* personalizados al formato de columnas estándar de GLUE.

Resumiendo, adaptaremos el código de GLUE para entrenar modelos Transformers con la librería HuggingFace.

Pipelines

Si queremos evaluar los modelos entrenados, primero necesitamos generar predicciones para después compararlas con el corpus original y aplicar métricas de evaluación sobre los resultados (F1, Precisión, Cobertura, etc.).

Antes de realizar las evaluaciones, necesitamos poner en marcha los modelos y generar predicciones. Para poder utilizar los modelos entrenados utilizamos la herramienta de *Pipelines* de HuggingFace's Transformers, mediante el cuál se conecta una red de procesos de datos en serie con los que, dado un modelo y un texto cualquiera, somos capaces de obtener una predicción en forma de etiqueta como salida. Mediante el uso de los *Pipelines* podemos coger cada tuit, predecir una etiqueta y almacenar ambos en ficheros de predicciones en formato .tsv (formato de fichero con valores separados por tabuladores).

4.3.3. API de Twitter

Tras analizar las herramientas necesarias para entrenar y entrenar los modelos de clasificación de texto, a continuación analizaremos aquellas destinadas a la recopilación del corpus.

Twitter ofrece de manera gratuita una serie de herramientas para el uso de sus datos bajo unos términos de uso. El uso que le daremos a la API será orientado al estudio y la investigación, por lo que Twitter nos lo pone relativamente fácil y solo tendremos que realizar los siguientes pasos:

1. Crear de una cuenta de Twitter.
2. Darnos de alta en la web enfocada a desarrolladores que proporciona Twitter [5].
3. Dar de alta una aplicación.

Una vez completados estos pasos, Twitter nos proporcionará un total de 4 claves secretas, intransferibles y únicas para cada aplicación. Gracias a ellas podremos utilizar paquetes como tweepy (python) para la gestión de la API.

Tweepy

La herramienta utilizada para la recolección de tuits es el paquete *Tweepy* [6]. A su vez, utilizamos dos de las herramientas que ofrece la librería: *Streaming* y *Cursor*. Para la recopilación de tuits se propone inicialmente utilizar la herramienta de *Streaming*. Mediante esta herramienta, se recogen todos los tuits que cumplen con los criterios de búsqueda publicados a partir del inicio del proceso de recolección. Sin embargo, la escasa cantidad de tuits publicados durante las horas en las que se mantiene activa la búsqueda resulta ser insuficiente. Por ello, se recurre a la herramienta *Cursor*, con la cual se puede acceder a los tweets ya publicados y almacenados en los servidores de Twitter. Esta herramienta permite recoger tuits publicados en los últimos 7 días.

[5] https://developer.twitter.com/en
[6] https://www.tweepy.org/

4.4. Evaluación

Para poder evaluar nuestros sistemas, utilizaremos el código proporcionado por HatEval para la competición. La evaluación realiza métricas estándar en éste tipo de modelos:

- Accuracy o exactitud: se trata de la fracción de predicciones que el modelo realizó correctamente. Se expresa de la siguiente forma:

$$Exactitud = \frac{Predicciones \quad correctas}{Total \quad de \quad predicciones}$$

- Precisión: expresa la proporción de predicciones afirmativas realizadas correctamente respecto al total de predicciones afirmativas realizadas en total.

$$Precisión = \frac{Verdaderos \quad positivos}{Total \quad de \quad predicciones \quad positivas}$$

- Recall o cobertura: expresa la proporción del número de etiquetas predichas correctamente respecto al total de respuestas correctas posibles

$$Cobertura = \frac{Predicciones \quad realizadas \quad correctamente}{Total \quad de \quad etiquetas \quad en \quad el \quad Gold \quad Standard}$$

El Gold Standard es un conjunto de datos que asumimos que está bien etiquetado.

- Valor F1 (macro): se utiliza para medir la precisión total. Se trata de una media armónica que combina los valores de la precisión y de la cobertura.

$$F1(macro) = 2 * \frac{Precisión * Cobertura}{Precisión + Cobertura}$$

Toda la evaluación es realizada mediante el código que proporciona la misma competición[7] con el fin de que cualquier persona interesada en participar en el desafío pueda evaluar su sistema fácilmente.

[7] https://github.com/msang/hateval/tree/master/SemEval2019-Task5/evaluation

5. CAPÍTULO

Experimentación

En este apartado se describen los experimentos llevados a cabo a lo largo del proyecto. En primer lugar está el **diseño de experimentos**, donde se relatan las decisiones tomadas, hiperparámetros utilizados para el entrenamiento de modelos, plataformas utilizadas para el desarrollo, etc.

Después, se analiza la experimentación con modelos entrenados con corpus HatEval, el *dataset* traducido y el corpus *Behagune* de polaridad.

Finalmente se hace un balance general de la evaluación de los modelos, comparando los mismos y realizando un análisis de errores.

5.1. Diseño de los experimentos

5.1.1. Esquema de experimentación

Los pasos a seguir en la experimentación han sido los siguientes:

1. A modo de baseline, entrenar un modelo en inglés y otro en castellano para la clasificación de texto mediante fastText. Se tratan de clasificadores lineales entrenados mediante aprendizaje supervisado.

2. Una vez establecido este límite inferior, se pasa al entrenamiento de modelos con

arquitectura Transformers con la librería de HuggingFace. Estos modelos serán versiones *Fine-tuneadas* del modelo pre-entrenado XLM-RoBERTa.

3. Etiquetar el corpus nativo de tuits en euskera con un modelo multilingüe pre-entrenado. En caso de obtener unos resultados adecuados en el corpus de tuits nativo, se pasaría a entrenar un nuevo modelo con ese corpus en euskera. En caso contrario, se obtiene un *dataset* en euskera mediante la traducción del *dataset* HatEval y se entrena un nuevo modelo con ese nuevo *dataset*.

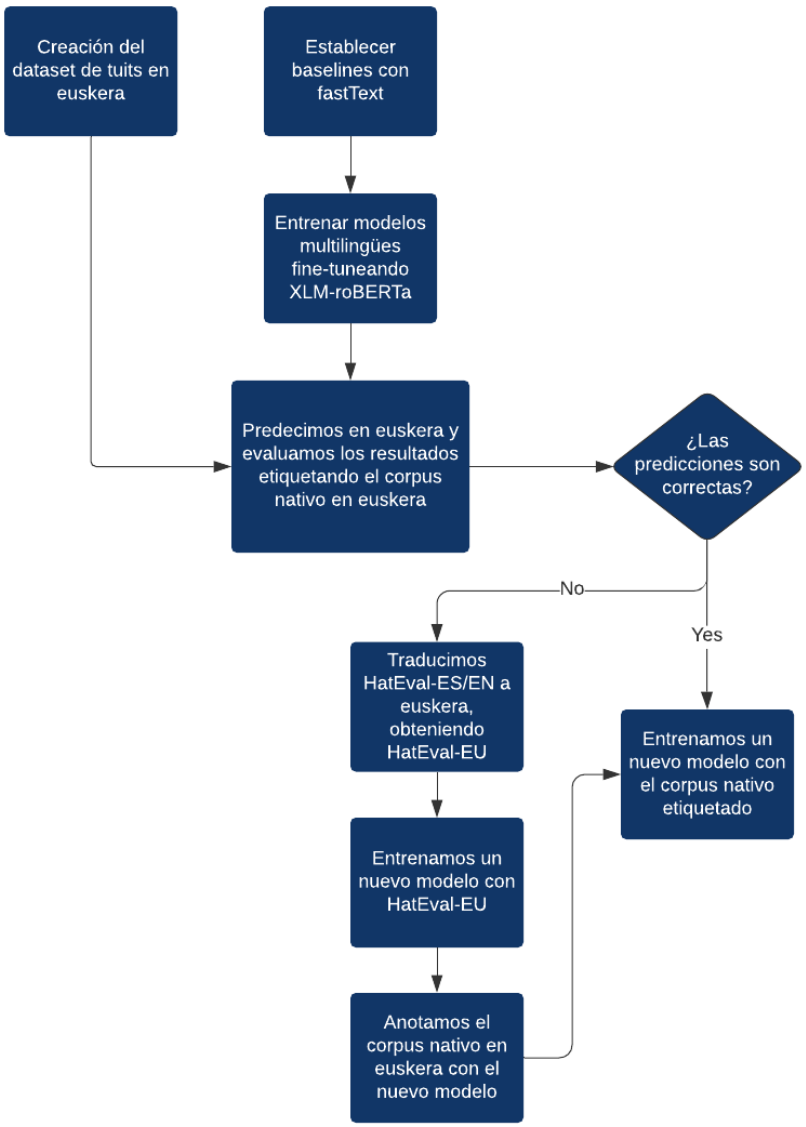

Figura 5.1: Diagrama de flujo de la toma de decisiones.

5.1.2. Configuración de fastText

Como se ha explicado previamente, fastText es una herramienta actualmente muy popular en el entrenamiento de modelos para la clasificación e ideal para construir unas bases de las que partir. Esta es la configuración utilizada para el entrenamiento de modelos:

Comandos e hiperparámetros

En nuestro caso, haremos uso de únicamente tres comandos:

1. *Supervised*: nos permitirá crear los modelos mediante aprendizaje supervisado (esto es, sobre corpus ya etiquetado con las etiquetas Odio/NoOdio). Necesitaremos siempre un archivo de entrenamiento en formato tsv (valores separados por tabulación) en formato fastText y un nombre para el modelo resultado. Además, podemos personalizar el entrenamiento del modelo a nuestro gusto con una serie de hiperparámetros que determinan las características del *Stochastic Gradient Descent* (SDG). El SDG es un algoritmo de optimización iterativo que, a cada iteración, actualiza los valores de la red neuronal en base a los datos obtenidos del *dataset* para así refinar y entrenar el modelo. En nuestro caso, utilizamos los siguientes parámetros:

   a) Learning rate (-lr): este parámetro [0,1] representa la varianza en cada iteración de los pesos de una red neuronal durante su entrenamiento. Es decir, determina la rapidez con la que el modelo aprende un problema. Cuanto más cercano a 1 sea el LR, más variarán los pesos en la red neuronal, dando lugar a un aprendizaje más rápido pero también más abrupto. En cambio, cuando más cercano a 0 sea el LR, los pesos variarán en menor medida, haciendo que en cada iteración el modelo varíe menos pero de manera más prudente.

   b) epoch (-epoch): determina el número de veces que se recorre el *dataset* durante el entrenamiento.

   c) N-gramas (-wordNgrams): un N-grama representa una cadena de N palabras seguidas en un texto. Por ejemplo, “Me llamo Pablo” representa un 3-grama.

   d) Pre-trained vectors (-pretrainedVectors *«vector.vec»*): Entrenamos también el modelo con uno de los 157 vectores de palabras que ofrece fastText, que fueron previamente entrenados por la misma librería. Cada uno de esos vectores de palabras está entrenado en un idioma distinto y no son más que representaciones vectoriales de las palabras que componen cada uno de los idiomas.

Este proceso de entrenamiento supervisado fue realizado tanto en castellano como en inglés, por lo que el proceso se realizó dos veces de manera casi idéntica, dando como fruto dos modelos

2. test: Una vez creado el modelo, fastText nos permite evaluar de manera sencilla el funcionamiento del mismo. Para ello, sólo hará falta utilizar el propio modelo y la partición de prueba del *dataset* de Hateval. El programa nos devolverá dos valores. Por un lado nos devuelve la Precisión, que no es más que el número de etiquetas correctas entre todas las predichas por el modelo. Por otro lado, nos devuelve la cobertura, que representa, de entre todas las etiquetas reales, el número de etiquetas que se han predicho correctamente. Para interpretar los resultados, hay que recordar que estamos trabajando con una clase binaria (Odio, ¿sí o no?).

3. predict: nos permitirá utilizar estos modelos para predecir etiquetas sobre un *dataset*. En concreto, la partición test. Lo utilizaremos para evaluar los modelos con el código de evaluación de HatEval.

Elección de hiperparámetros

La elección de hiperparámetros se realiza sobre el *subset* de desarrollo (.dev) del *dataset* con el que entrenamos los modelos. Tanto para el modelo en inglés como el de castellano la configuración de hiperparámetros fue idéntica:

1. Learning-rate: 1.0
2. Epochs: (10,15,20,25)
3. n-gramas: 2

En el esquema de hiperparámetros utilizamos siempre un valor 1.0 de *Learning-rate*, 2-gramas, y variamos entre 10,15,20 y 25 epochs.

Adicionalmente, utilizamos vectores de palabras pre-entrenados para el entrenamiento los de modelos de clasificación de texto[1]. Los vectores de palabras se tratan de palabras o frases del lenguaje natural representadas como vectores de números reales y nos ayudarán a obtener mejores modelos de clasificación de texto. fastText tiene disponibles estos vectores de palabras en 157 idiomas distintos. Nosotros utilizamos únicamente 2 de ellos, uno

[1] https://fasttext.cc/docs/en/crawl-vectors.html

en inglés y otro en castellano. Utilizamos éstos vectores de palabras como un hiperparámetro más en el entrenamiento de los modelos de clasificación de texto: es decir, además de especificar el *Learning-rate*, el número de *epochs* y los n-gramas, especificamos también si queremos utilizar vectores de palabras mediante el hiperparámetro *wordVectors*, seguido del vector de palabras a utilizar.

5.1.3. Configuración de HuggingFace's Transformers

Si bien sería posible realizarlo de forma local, finalmente hacemos uso de Google Colaborate para el entrenamiento de modelos mediante HuggingFace's Transformers. Esta herramienta nos permite utilizar una tarjeta gráfica de gran potencia de forma gratuita en la nube (la plataforma nos ofrece la tarjeta NVIDIA Tesla K80 (12 GB)). Por tanto, todo el código relacionado con el entrenamiento de Transformers se desarrolla en esa plataforma y todos los ficheros necesarios se almacenan en Google Drive.

Para el correcto entrenamiento de los modelos, la combinación de hiperparámetros es crucial. Para poder entrenar nuestos modelos con cualquier corpus, ha sido necesario adaptar el código de GLUE (benchmark), que de forma predeterminada únicamente nos permite realizar Fine-Tune con uno de sus *datasets*. A continuación se muestra un ejemplo de Fine-Tune con el código de GLUE y uno de sus *datasets* disponibles: Standford Sentiment Treebank o SST-2. Utilizamos el *dataset* SST-2 de porque se trata de un corpus para la clasificación de texto y análisis del sentimiento, similar a nuestra tarea de detección de odio.

Comandos e hiperparámetros

Para Fine-Tunear XLM-RoBERTa con el *dataset* SST-2 mediante el benchmark GLUE elegimos la siguiente combinación de parámetros:

```
$ python examples/run_glue.py --model_type xlm-roberta --model_name_or_path xlm-roberta-base --
   task_name SST-2 --do_train --do_eval --data_dir examples/glue/glue_data/SST-2 --
   max_seq_length 128 --per_gpu_train_batch_size 32 --learning_rate 2e-5 --num_train_epochs 3.0
    --output_dir /tmp/SST-2/
```

- –model_type: xlm-roberta. se elige un modelo entre los ofrecidos por la librería de transformers. En este caso, XLM-roBERTa.
- –model_name_or_path: xlm-roberta-base. De nuevo, elegimos xlm-RoBERTa como nuestro modelo a entrenar
- –task_name: SST-2. la tarea sobre la que queremos entrenar el modelo. En este caso, SST-2.
- –do_train: indicamos que queremos entrenar el modelo
- –do_eval: indicamos que queremos evaluar el modelo
- –data_dir: glue_dataSST-2. ubicación del *dataset* SST-2
- –max_seq_length: 128. la longitud máxima de las secuencias es de 128
- –per_gpu_train_batch_size: 32. Se limita la cantidad de datos con la que se entrena.
- –learning_rate: 2e-5. Se indica la velocidad a la que la red neuronal aprende.
- –num_train_epochs 3.0. Indica el número de veces que se recorre el *dataset* durante el entrenamiento del modelo.
- –output_dir /tmp/SST-2/. El directorio en el que se guarda el modelo una vez entrenado.

Elección de hiperparámetros

A la hora de entrenar los modelos Transformers, la configuración de hiperparámetros fue la siguiente:

1. Learning-rate: 0,00002 (2E-5)
2. Epochs: (3,5,10)
3. batch-size: 32
4. sequence-length: 128

Los hiperparámetros se seleccionan sobre el *subset* de desarrollo (.dev) del *dataset* con el que entrenamos los modelos. En el proceso de entrenamiento de modelos el único hiperparámetro que varía son la cantidad de *epochs*. Como veremos en los resultados de la evaluación de los modelos (ver Tabla 5.3), la diferencia de rendimiento entre los modelos entrenados con 3 *epochs* y los entrenados con 10 es muy grande.

5.2. Detección de expresiones de odio en inglés y castellano sobre corpus HatEval

5.2.1. Baseline: fastText

Se entrenando dos modelos para la clasificación de texto mediante fastText con los corpus de HatEval en castellano e inglés. Utilizamos la evaluación de HatEval para medir la calidad de los modelos entrenados. Estos son los resultados:

Modelo entrenado con HatEval-ES			
F1-score	Precisión	Cobertura	Accuracy
0,7043	0,7041	0,7046	0,7131
Modelo entrenado con HatEval-EN			
F1-score	Precisión	Cobertura	Accuracy
0,4879	0,5772	0,5544	0,5083

Tabla 5.1: Evaluación de modelos entrenados en fastText

Como hemos visto en secciones anteriores, el *dataset* en inglés contiene graves errores de etiquetado. Este problema es claramente visible con echar un vistazo a los resultados, donde vemos que el valor F1 es muy superior en el modelo en castellano que en el de inglés.

5.2.2. Transformers

En total se entrenan 18 modelos Transformers de la siguiente manera: partimos de los 3 *datasets* HatEval (en inglés, en castellano y la mezcla de ambos) en sus dos versiones distintas (preprocesados y no-preprocesados). Con cada uno de estos seis *datasets* se entrena un modelo variando el hiperparámetro 'epoch' a 3, 5 y 10 epoch.

Evaluación de los modelos Transformers

Para realizar la evaluación de modelos se han seguido los siguientes pasos:

1. Escoger el modelo que queremos evaluar (en nuestro caso, los 18 modelos que hemos entrenado).
2. Escoger un *dataset* de referencia con el que realizar la evaluación (únicamente utilizamos la partición de prueba del *dataset*, .test). En nuestro caso utilizamos dos *datasets*: HatEval-ES y HatEval-EN. Ambos tienen una estructura de dos columnas, siendo la primera para el texto del tuit y la segunda para la etiqueta del propio tuit.
3. Separamos los tuits de su etiqueta, guardando las etiquetas originales para más tarde y quedándonos únicamente con el texto de los tuits de los *datasets* de referencia.
4. Dado el modelo a evaluar y la lista de tuits de los *datasets* de referencia, se realizan predicciones que tienen como salida una etiqueta por cada tuit.
5. Finalmente se comparan las etiquetas originales del *dataset* de referencia con las nuevas etiquetas. La evaluación se realiza mediante el código proporcionado por la competición HatEval en el que se realizan **métricas de valor F1, Precisión, Cobertura y Exactitud**.

En total se realizan **36 evaluaciones**, dos (en inglés y castellano) por cada uno de los **18 modelos**.

5.2.3. Resultados y análisis

Todos los resultados obtenidos se recogen en dos tablas: la Tabla 5.2 muestra los resultados de la evaluación (rendimiento) de los modelos entrenados con *datasets* **no-preprocesados**,

mientras que la Tabla 5.3 muestra los entrenados con *datasets* **preprocesados**. Por otro lado, ambas tablas están divididas en dos partes. En la primera mitad se muestra la evaluación realizada con el *dataset* HatEval-ES (castellano) mientras que la segunda mitad se muestra la realizada con HatEval-EN (inglés). Finalmente, la Tabla 5.4 recoge los mejores modelos entrenados.

Una de las conclusiones a la que llegamos es que el comportamiento de los modelos cambia drásticamente dependiendo de si los *datasets* con los que se entrenan están preprocesados o no. En lineas generales el comportamiento de los modelos entrenados con *datasets* **preprocesados es considerablemente superior al de los no-preprocesados**, ya que si se compara cada celda individualmente se ve que los valores en la tabla de los no-preprocesados (5.2) son inferiores que los valores de la tabla de los preprocesados (5.3). Por ejemplo el modelo de 5 epochs eternado en inglés y castellano obtiene un valor F1 de 0,7774 en su versión preprocesada, que es superior al valor F1 de 0,7642 obtenido en su versión no-preprocesada Además, la mezcla de los *datasets* de inglés y castellano para el Fine-Tune de los modelos resulta ser una buena idea, ya que son los modelos que obtienen los mejores resultados en casi todos los casos de la evaluación: como se muestra en la Tabla 5.4, 3 de los mejores modelos están entrenados en los dos idiomas (inglés y castellano).

Respecto a la combinación de hiperparámetros escogida, los modelos entrenados con 5 y 10 epochs son superiores a los de 3 epochs. Por tanto, llegamos a la conclusión de que para entrenar modelos con *datasets* del tamaño de los utilizados es recomendable que durante el entrenamiento se recorra más de 3 veces el *dataset*.

Además, podemos observar que los modelos se comportan mucho mejor al evaluarlos con el *dataset* en castellano que en el de inglés. Esto está relacionado con los problemas del *dataset* HatEval-EN analizados en el capítulo 4.1.1. La diferencia de calidad entre ambos corpus se hace aún más notable al observar que los modelos entrenados en ambos idiomas obtienen una evaluación superior en inglés respecto a los modelos entrenados únicamente en este idioma. Por ejemplo, en la Tabla 5.3 observamos que en la segunda mitad (destinada a la evaluación en inglés de los modelos) el mejor modelo es el entrenado con 10 epochs en inglés y castellano, con un valor F1 de 0,6058 respecto al entrenado en 10 epochs únicamente en ingles con un valor F1 de 0.5787. Incluso el modelo entrenado únicamente en castellano con los mismos epochs consigue prácticamente el mismo rendimiento, con un valor F1 de 0.5775. Estas diferencias se se acentúan en la tabla de no-preprocesados (5.2).

Como consecuencia de este proceso de entrenamiento y evaluación de modelos transformers, concluimos que el modelo entrenado con el corpus en ambos idiomas, preprocesado y con 10-epochs de entrenamiento es el que más favorecido sale de la evaluación. Para la evaluación en castellano, éste modelo obtiene un valor F1 de 0,7753; para la evaluación en inglés, obtiene un valor F1 de 0,6058.

Rendimiento de los modelos con HatEval-ES				
Modelos	F1-score	Precisión	Cobertura	Exactitud
3-epoch / eng+esp	0,7456	0,7542	0,7607	0,7462
5-epoch / eng+esp	0,7642	0,7712	0,7787	0,7650
10-epoch / eng+esp	**0,7719**	**0,7761**	**0,7845**	**0,7731**
3-epoch / eng	0,7456	0,7542	0,7607	0,7462
5-epoch / eng	0,6621	0,6978	0,6609	0,6962
10-epoch / eng	0,6324	0,6816	0,6357	0,6775
3-epoch / esp	0,7551	0,7601	0,7679	0,7562
5-epoch / esp	0,7693	0,7690	0,7771	0,7718
10-epoch / esp	0,7553	0,7552	0,7630	0,7581
Rendimiento de los modelos con HatEval-EN				
Modelos	F1-score	Precisión	Cobertura	Exactitud
3-epoch / eng+esp	0,4573	0,6461	0,5659	0,5043
5-epoch / eng+esp	0,4815	0,6539	0,5789	0,5203
10-epoch / eng+esp	0,4765	0,6567	0,5772	0,5176
3-epoch / eng	0,4443	0,6528	0,5615	0,4976
5-epoch / eng	0,4812	0,6723	0,5833	0,5230
10-epoch / eng	0,4925	0,6665	0,5874	0,5293
3-epoch / esp	0,5646	0,6482	0,5867	0,6370
5-epoch / esp	0,5403	0,6761	0,5780	0,6363
10-epoch / esp	**0,5984**	**0,6789**	**0,6129**	**0,6596**

Tabla 5.2: Modelos HatEval sin preprocesar

Rendimiento de los modelos con HatEval-ES				
Modelos	F1-score	Precisión	Cobertura	Exactitud
3-epoch / eng+esp	0,7628	0,7693	0,7770	0,7637
5-epoch / eng+esp	**0,7774**	**0,7784**	**0,7871**	**0,7793**
10-epoch / eng+esp	0,7753	0,7778	0,7866	0,7768
3-epoch / eng	0,6545	0,6551	0,6541	0,6662
5-epoch / eng	0,6607	0,6599	0,6621	0,6687
10-epoch / eng	0,6294	0,6309	0,6286	0,6437
3-epoch / esp	0,7534	0,7531	0,7607	0,7562
5-epoch / esp	0,7476	0,7529	0,7604	0,7487
10-epoch / esp	0,7728	0,7721	0,7801	0,7756
Rendimiento de los modelos con HatEval-EN				
Modelos	F1-score	Precisión	Cobertura	Exactitud
3-epoch / eng+esp	0,5799	0,6817	0,6375	0,5930
5-epoch / eng+esp	0,5820	0,6782	0,6374	0,5940
10-epoch / eng+esp	**0,6058**	**0,6893**	**0,6545**	0,6143
3-epoch / eng	0,5705	0,6761	0,6305	0,5850
5-epoch / eng	0,5768	0,6817	0,6358	0,5906
10-epoch / eng	0,5787	0,6728	0,6337	0,5906
3-epoch / esp	0,5967	0,6394	0,6044	0,6433
5-epoch / esp	0,5915	0,6511	0,6035	0,6470
10-epoch / esp	0,5775	0,6778	0,5996	**0,6510**

Tabla 5.3: Modelos HatEval preprocesados

SIN PREPROCESAR				
Rendimiento de los modelos con HatEval-ES				
Modelos	F1-score	Precisión	Cobertura	Exactitud
10-epoch / eng+esp	0,7719	0,7761	0,7845	0,7731
Rendimiento de los modelos con HatEval-EN				
Modelos	F1-score	Precisión	Cobertura	Exactitud
10-epoch / esp	0,5984	0,6789	0,6129	0,6596
PREPROCESADOS				
Rendimiento de los modelos con HatEval-ES				
Modelos	F1-score	Precisión	Cobertura	Exactitud
5-epoch / eng+esp	0,7774	0,7784	0,7871	0,7793
Rendimiento de los modelos con HatEval-EN				
Modelos	F1-score	Precisión	Cobertura	Exactitud
10-epoch / eng+esp	0,6058	0,6893	0,6545	0,6143

Tabla 5.4: Mejores modelos HatEval

Comparación de resultados con respecto al desafío HatEval

Si comparamos los resultados obtenidos con nuestros modelos y el de los mejores sistemas presentados a la competición de HatEval en 2019 (Tabla 5.5), observamos que obtiene unos resultados superiores en la competición en castellano, mientras que en la competición en inglés no consigue superar el resultado del equipo Fermi.

Tarea de evaluación en castellano	
Models	F1-score
Atalaya Team	0,7300
MineriaUNAM Team	0,7300
Media de la competición	0,6821
(Nuestro)fastText	0,7043
(Nuestro)10-epoch / eng+esp	**0,7753**
Tarea de evaluación en inglés	
Models	F1-score
Fermi Team	**0,6510**
Media de la competición	0,4483
(Nuestro)fastText	0,4879
(Nuestro)10-epoch / esp	0,6058

Tabla 5.5: Comparación entre los sistemas presentados a HatEval y los nuestros

5.3. Detección de expresiones de odio en euskera sobre corpus traducido y nativo

5.3.1. Experimentación

Como ya hemos visto, uno de los objetivos del proyecto es el de conseguir modelos capaces de detectar discurso de odio en euskera. Con los modelos multilingües Transformers descritos en el apartado 5.2.2 es posible detectar mensajes de odio, pero el resultado obtenido posiblemente no sea el mejor. Aunque sean modelos pre-entrenados en euskera y, por tanto, puedan realizar predicciones en éste idioma, lo ideal sería obtener un modelo entrenado directamente en corpus de discurso de odio en euskera y no en inglés y castellano como ocurre con los que hemos entrenado.

Con el fin de evaluar los modelos multilingües entrenados y, a su vez, crear nuevos modelos, se propone la creación de los siguientes *datasets* de corpus de odio en euskera:

1. Un corpus recogido manualmente de la comunidad de Twitter en euskera

2. Un corpus traducido de HatEval-ES (en castellano), al euskera

Corpus nativo de tuits en euskera

Características generales:

El *dataset* nativo en euskera contiene un total de 11.272 tuits formados por 149.754 palabras y publicados por 2.861 usuarios diferentes. Todos los tuits fueron recogidos en el periodo entre el 5 de abril de 2020 a las 15:00 y el 28 de mayo del mismo año a las 08:45. En la Tabla 5.6 se recogen distintas características del *dataset*.

Características del *dataset* nativo en euskera	
Usuarios más frecuentes del *dataset*	Usuarios más mencionados por otros usuarios
@euskaltel @EstepanPlazaola @GariGaraialde @Venezuelatierr1 @EstebanRG_ @AinhoaRoitegi @Deteibols @omijmitxelena @argindar @erramunms	@euskaraldia @VG_Policiaren @osakidetzaEJGV @berria @EHIkastolak @LABsindikatua @YouTube @Gob_eus @gob_na @LeakoHitza
Localizaciones más frecuentes del *dataset*	Tendencias más frecuentes del *dataset*
Euskal Herria Euskadi Navarra - Nafarroa Basque Country Tolosa Bermeo, España Donostia Hondarribia Gipuzkoa *Sin ubicación*	#covid19 #koronabirusa #etxealditikeuskaraldira #quedateencasa #aberrieguna2020 #covid-19 #aberrieguna #bideoa #etxeangeratu #COVID

Tabla 5.6: Características del *dataset* nativo en euskera

Como ya se ha explicado en varias ocasiones en éste documento, el *dataset* nativo de tuits en euskera apenas recoge tuits con discurso de odio y las características que se extraigan del corpus no tendrán relación con ese tipo de mensajes. Por tanto, la lista de usuarios más frecuentes que recoge la Tabla 5.6 no representa una lista de usuarios que propagan discurso de odio, sino de usuarios que son frecuentes en la red social (ya sean periodistas, medios de comunicación, etc.).
Por otro lado, las localizaciones recogidas en la tabla pueden no representar fielmente la realidad, ya que los usuarios de Twitter pueden escribir su ubicación a mano y pueden elegir un lugar que no corresponde a su ubicación real. También es posible que existan ubicaciones repetidas, pues un usuario puede elegir por ejemplo 'Donostia', otro 'San Sebastián' y serán detectados como lugares distintos aunque en realidad sean el mismo.
Por último, es importante destacar que, pese a recoger las tendencias (*hashtags*) en la tabla, el *dataset* con el que se entrenan los modelos en euskera es una versión preprocesada del mismo y las tendencias se eliminaron para entrenar los modelos.

Recolección:

La recogida de datos tiene lugar en el periodo entre el 13 de abril y el 28 de mayo de 2020. Durante estos dos meses, se realizan recogidas de tuits en periodos de 2 a 7 días (es decir, en cada recolección se recogen los tuits publicados en los últimos 2-7 días). Debido al alto movimiento de tuits durante estos dos meses (que coinciden con los dos primeros meses de la pandemia del COVID-19), el criterio de búsqueda de tuits cambió cada semana. Para obtener una visión general de los temas trending en la comunidad de Twitter en euskera, se utiliza la herramienta umap [2], mediante la cual podemos realizar un análisis exhaustivo de las tendencias de cada día en cuanto a temas más tratados, hashtags más utilizados, usuarios más relevantes, etc. Algunos de los hashtags polémicos y controversiales utilizados para la búsqueda de tuits fueron los relacionados con la pandemia (#covid, #koronabirusa, #etxeangeratu...), otros relacionados con protestas en sectores concretos como el estudiantil (#escuchanosupvehu) y otros casos mediáticos (#zaldibarargitu, #altsasu...).

[2] https://umap.eus/

Estructura:

Los tuits fueron almacenados en archivos *column separated value* o .csv con la siguiente estructura:

- username (@)
- ubicación
- user_id
- tweet_id
- texto
- fecha

Anotación:

Hasta este momento contamos con un corpus nativo en euskera sin etiquetar, así como unos modelos multilingües especializados en discurso de odio cuyo objetivo sería el de realizar la anotación de nuestro *datasets* de tuits en euskera. Sin embargo, al principio del proyecto planteamos la posibilidad de que el corpus en euskera no contuviera la suficiente carga de odio y que por tanto las etiquetas entre Odio / No-odio estuvieran desbalanceadas. Como veremos más adelante, éste es el caso.

Además de los *pipelines* disponibles en la librería Transformers de HuggingFace (más sobre pipelines en 4.3.2), para etiquetar el **corpus nativo de tuits en euskera** utilizamos el modelo Transformers entrenado con 10-epoch y corpus en inglés y castellano preprocesado (más en la Tabla 5.3). Por cada tuit del corpus nativo en euskera, se realizará una predicción con nuestro modelo y se le añadirá como etiqueta al *dataset* que indica si ese tuit contiene algún mensaje de odio o no.

Hay que tener en cuenta que el modelo utilizado para etiquetar el *dataset* no fue entrenado exclusivamente en euskera. Es decir, nuestro modelo Transformer parte de un modelo preentrenado en 100 idiomas (entrenado en euskera) pero el entrenamiento posterior (Fine-Tune) fue realizado con el *dataset* HatEval en inglés y castellano. Al proceso de realizar predicciones en un corpus desconocido por el modelo y en un idioma en el que no fue entrenado, se le llama *Zero-Shot learning*.

Corpus traducido al euskera: HatEval-EU

Para **traducir a euskera el corpus de HatEval** en castellano (HatEval-ES) utilizamos la librería Transformers de HuggingFace, la cual tiene disponible Opus-MT [3], traductores automáticos basados en redes neuronales. En concreto utilizaremos el modelo 'opus-mt-es-eu' para la traducción de castellano a euskera.

5.3.2. Resultados y análisis

Análisis del *dataset* nativo

Analizando el *dataset* nativo etiquetado obtenido, observamos que de un total de 11.521 tuits, solo 442 han sido detectados como odio (únicamente un 3,8 % de los tuits). Ésto se puede deber a dos factores:

1. El *dataset* está limpio de odio y, por tanto, no es válido para el entrenamiento de un modelo como el que buscamos.
2. El modelo Transformer utilizado para etiquetar el corpus no es lo suficientemente robusto y no ha podido detectar correctamente el odio en los tuits.

Por tanto, desechamos la idea de generar un sistema con este *dataset*, pues llegamos a la conclusión de que cuenta con un desbalance de etiquetas demasiado grande como para entrenar un modelo de calidad con él.

Análisis del *dataset* traducido HatEval-EU

En general, consideramos que la traducción del *dataset* HatEval-EU es bastante robusta. Los tuits traducidos mantienen en su mayoría el significado original y al partir de un *dataset* como HatEval-ES, sabemos que la carga de odio es predominante también después de la traducción: el corpus HatEval-ES contenía un total de 6.536 tuits, de los cuales 2.713 contenían odio. Por tanto, la traducción HatEval-EU mantiene dicha proporción de odio en el *dataset* (un 41,5 %, una gran mejora respecto al 3,8 % del *dataset* nativo). Sin embargo, la traducción realizada por el sistema Opus-MT es, en ocasiones, demasiado literal.

[3] https://huggingface.co/Helsinki-NLP/opus-mt-es-eu

Como podemos ver en los ejemplos recogidos en la Tabla 5.7, algunos de los tuits traducidos pierden la connotación inicial. Por ejemplo, en el primer tuit la expresión "seguro que tienes maridos"del primer tuit de la tabla es traducida como "senar seguruak dituzu", es decir, "tienes maridos seguros". Ocurre algo similar con el segundo tuit, donde la traducción es simplemente errónea y el tuit pierde el significado totalmente. Este tipo de traducciones empeorarán el modelo entrenado con este *dataset*, ya que la traducción *Galdu egingo gara* mantiene la etiqueta de odio del tuit original aún si ya no tiene ese significado.

A estos casos se unen también las traducciones curiosas de los tuits de la figura 5.2, donde además de errónea (o no realizada), la traducción añade emoticonos musicales sin ningún motivo.

YA CALLATE PERRA PLIS	♪ Ya callate perra PLIS
Que temazo arabe dios mio	♪ Ai ene, ene Jainkoa! ♪

Figura 5.2: Curiosidades de la traducción de HatEval-EU

Aún así, el sistema Opus-MT ha realizado traducciones realmente versátiles. Por ejemplo, en el tercer tuit de la Tabla 5.7 vemos que el sistema es capaz de traducir insultos inventados en castellano.

HatEval-ES	HatEval-EU
Tu eres puta..tienes maridos seguro	Zu puta zara... senar seguruak dituzu..
callate que mañana nos vemos perra	Galdu egingo gara
incompatible con ser moromierda o musulmono	moromierra edo musulmonoa izatea bateraezina da

Tabla 5.7: Traducciones erróneas de HatEval-EU

Evaluación de modelos creados con el *dataset* HatEval-EU

Los modelos creados con el *dataset* traducido HatEval-EU han sido entrenados de la misma forma que los demás modelos Transformers. Es decir, se ha partido del modelo pre-entrenado XLM-RoBERTa y se ha *Fine-Tuneado* con el *dataset* HatEval-EU. Sin embargo, se ha cambiado el esquema de selección e hiperparámetros: al evaluar los modelos Transformers (más en 5.4), observamos que los modelos entrenados con 3 *epochs* obtienen los peores resultados en todos los apartados. Por tanto, esta vez cambiaremos los diferentes *epochs* con los que entrenar pasando de *3, 5, y 10* a *5, 10 y 15*.

En la Tabla 5.8 se recogen los resultados de la evaluación y se obtienen de la misma forma que con los anteriores modelos Transformers: dados el modelo HatEval-EU a evaluar y un fichero de prueba (HatEval-EU.test) se comparan las etiquetas predichas por el modelo con las etiquetas originales del fichero de prueba.

Rendimiento de los modelos entrenados con HatEval-EU				
Modelos a evaluar	F1-score	Precisión	Cobertura	Exactitud
HatEval-EU / 5-epoch	0,6451	0,6523	0,6565	0,6463
HatEval-EU / 10-epoch	**0,6657**	**0,6700**	**0,6752**	**0,6675**
HatEval-EU / 15-epoch	0,6416	0,6501	0,6539	0,6425

Tabla 5.8: Modelos entrenados con HatEval-EU

Observamos que el mejor modelo de los 3 entrenados es el entrenado con 10 *epochs*, pese a intentar obtener un mejor valor utilizando 15 *epochs*.

5.4. Identificación de la polaridad de opiniones y su relación con expresiones de odio

5.4.1. Experimentación

Para complementar la identificación de odio de los *datasets* con los que trabajamos, medimos también la polaridad de las opiniones de lo usuarios de Twitter. Éste era uno de los objetivos del proyecto, pues puede enseñarnos sobre la relación entre el odio y la polaridad y así ver hasta qué punto, por ejemplo, los mensajes de odio recogidos en los corpora tienen tendencia negativa o positiva. La medición de la polaridad puede ayudarnos tam-

bién a determinar la calidad de la traducción del *dataset* HatEval-EU, pues si ésta ha sido correcta, la polaridad del *dataset* HatEval-ES y HatEval-EU no debería ser distinta.

Los *datasets* y sus particiones

Para poder realizar predicciones de polaridad en base a los tuits, necesitamos modelos entrenados con corpus de polaridad. Como se explica en el apartado 4.1.3, utilizamos el corpus *Behagune* para la polaridad. Al contar con *datasets* en castellano y en euskera, el objetivo es conseguir modelos para cada idioma.

Los *datasets* originales *Behagune* no vienen divididos en las particiones clásicas para el entrenamiento de modelos, así que los dividimos en particiones de entrenamiento, desarrollo y prueba. Esta división en particiones se realiza dos veces por cada *dataset* en diferentes porcentajes. Es decir, cogemos los *datasets* behagune originales (en castellano y en euskera) y de cada uno obtenemos dos *datasets*: el primero se obtiene dividiendo el *dataset* en un 60 % para la partición de entrenamiento y un 20 % para las particiones de desarrollo y prueba. El segundo *dataset*, en cambio, se consigue dividiendo el *dataset* en un 80 % para la partición de entrenamiento y un 10 % para las particiones de desarrollo y prueba. La división de en particiones se realiza de manera completamente aleatoria.

Por tanto, obtenemos 4 *datasets*:

- *Dataset behagune* en castellano: 60 % train, 20 % dev, 20 % test
- *Dataset behagune* en castellano: 80 % train, 10 % dev, 10 % test
- *Dataset behagune* en euskera: 60 % train, 20 % dev, 20 % test
- *Dataset behagune* en euskera: 80 % train, 10 % dev, 10 % test

Entrenamiento de modelos

Con cada corpus entrenamos un modelo para la medición de polaridad, sumando un total de 4. Se realizan, una vez más, mediante la librería HuggingFace's Transformers y *Fine-Tuneando* el modelo pre-entrenado XLM-RoBERTa.

La configuración escogida para el entrenamiento de los modelos ha sido la que ha mostrado obtener los mejores resultados a lo largo del proyecto:

- Learning Rate: 2e-5
- Sequence Length: 128
- Batch Size: 32
- Epoch: 10

La evaluación de los modelos entrenados se recoge en la Tabla 5.9, donde en la primera columna se encuentran los nombres de los modelos, en la segunda el idioma del *dataset* con el que han sido entrenados y en la tercera la exactitud, obtenida ésta vez por la evaluación automática que realiza HuggingFace's Transformers.

Dataset	Idioma	Exactitud
Behagune 10epoch-ES (6-2-2)	Castellano	0,7852
Behagune 10epoch-ES (8-1-1)	**Castellano**	**0,8063**
Behagune 10epoch-EU (6-2-2)	Euskera	0,7291
Behagune 10epoch-EU (8-1-1)	**Euskera**	**0,7781**

Tabla 5.9: Datasets Behagune

Observamos que los *datasets* con la distribución 80-10-10 obtienen una mayor puntuación en Exactitud que los de distribución 60-20-20, por lo que utilizaremos los primeros para la medición de polaridad.

Proporción de odio y polaridad en *datasets* HatEval

Utilizamos los *datasets* HatEval-ES y HatEval-EU para probar nuestros modelos de polaridad. El objetivo de estas mediciones es el de investigar las relaciones entre los mensajes de odio y la polaridad de las opiniones. Para ello, ampliaremos los *datasets* HatEval-ES y HatEval-EU con una nueva columna: la de la polaridad. Por tanto, en estos nuevos *datasets* a cada tuit le corresponderá un etiqueta de odio (existe o no) y otra etiqueta de polaridad (positiva, neutra o negativa).

5.4.2. Resultados y análisis

Como ya se ha mencionado previamente, el *dataset* HatEval-EU es una traducción de HatEval-ES. En el caso ideal, al tratarse de una traducción, la polaridad debería ser exactamente la misma en ambos datasets. Es decir, que si cogemos la polaridad de un tuit

aleatorio del *dataset* en castellano y la comparamos con la polaridad de ese mismo tuit traducido en HatEval-EU, la polaridad debería ser la misma. Con el fin de comprobar hasta qué punto ha sido buena la traducción y los modelos de polaridad, comparamos la etiqueta de Odio de cada tuit con su etiqueta de polaridad.

Comparando las etiquetas predichas por los modelos en el *dataset* HatEval-EU con las de HatEval-ES observamos que la diferencia de etiquetas no es pequeña. De un total de 6.571 etiquetas, 4.780 son iguales y 1.791 son distintas. Como hemos dicho, en el caso ideal todas las etiquetas deberían ser igual iguales y, sin embargo, un 37,4 % de las etiquetas de polaridad son distintas. Ésto podría significar que los modelos entrenados no son lo suficientemente buenos, o que la traducción del *dataset* HatEval-ES para obtener HatEval-EU no fue buena. Sin embargo, es posible también que las variaciones de las etiquetas se deba a que la etiqueta de polaridad puede tomar 3 valores: positivo, neutro, negativo. De esta forma, es posible que algunas de las etiquetas positivas del *dataset* HatEval-EU se traten de etiquetas neutras en el *dataset* HatEval-ES, lo que sería alteración más leve que si se tratara de un cambio de neutro a negativo.

Para entender un poco más sobre la relación de odio y polaridad, obtenemos la coincidencia de cada una de las etiquetas de polaridad con las etiquetas de odio. Éstos resultados se recogen en la Tabla 5.10.

Observando los resultados obtenidos, vemos que existe una gran carga de polaridad negativa en ambos *datasets*, siendo un total de 5448 tuits en el *dataset* en castellano (83 % del corpus) y 4683 tuits en el *dataset* en euskera (71 %). Por otro lado, la polaridad positiva aumenta más del doble en el *dataset* en euskera respecto al de castellano. Finalmente, la carga de polaridad neutra apenas varía de un *dataset* a otro.

Por tanto, observamos que tras la traducción, el *dataset* en euskera disminuye su polaridad negativa y aumenta la positiva. La razón por la que esto ocurre no es segura: por un lado, es posible que se deba a que los modelos de polaridad no son perfectos y los errores cometidos en sus predicciones den lugar a esta diferencia de etiquetas. Por otro lado, es posible también que la manera de expresar negatividad en euskera sea distinta a la del castellano y que, por tanto, sea más fácil detectarlas en castellano que en euskera.

Por otro lado, la relación entre odio y polaridad muestra el siguiente comportamiento. En el *dataset* en castellano, de entre las etiquetas positivas un 30 % contienen también mensaje de odio. De entre las etiquetas negativas, como es de esperar, la cantidad de mensaje de odio es superior, llegando al 43 % de los tuits. En el *dataset* en euskera existe un escenario similar entre etiquetas positivas y negativas: entre las primeras, un 40 % de

Dataset HatEval-ES			
	POSITIVO	NEUTRO	NEGATIVO
Odio	145	206	2369
Sin odio	337	392	3119
Dataset HatEval-EU			
Odio	441	263	2016
Sin odio	657	524	2667

Tabla 5.10: Comparaciones entre odio y polaridad

los tuits contienen también mensajes de odio, mientras que en las etiquetas negativas un 43 % de los tuits contiene mensajes de odio.

6. CAPÍTULO

Conclusiones y trabajo futuro

Concluimos este documento con una breve reflexión de lo que ha supuesto el proyecto a nivel académico y personal.

6.1. Conclusiones

6.1.1. Objetivos cumplidos

A lo largo del desarrollo del proyecto se han ido cumpliendo los objetivos propuestos inicialmente. Por un lado, se ha realizado un análisis exhaustivo del panorama actual de la detección de odio mediante aprendizaje automático. Se ha obtenido un conocimiento de las distintas técnicas utilizadas para el Procesamiento del Lenguaje Natural que se ha aplicado para el desarrollo de sistemas multilingües capaces de detectar discurso de odio en varios idiomas, euskera incluído. Además, se ha construido un *dataset* de tuits en euskera que, si bien no contiene la carga de odio que buscábamos en un inicio, se ha utilizado para probar los modelos desarrollados y medir la interrelación del odio y la polaridad en los mensajes de los usuarios.

6.1.2. Reflexión personal

Este Trabajo de Fin de Grado (TFG) ha supuesto un punto de inflexión en mi carrera académica ya que me ha dado la motivación para seguir formándome en el campo del

aprendizaje automático y en concreto del Procesamiento del Lenguaje Natural.

A nivel personal, el desarrollo de este TFG ha sido muy satisfactorio. Por un lado, estoy satisfecho con la constancia de trabajo que ha requerido la creación de un *dataset* y la investigación para estar al día en las técnicas con las que trabajamos. Además, la organización del proyecto ha sido muy orgánica debido a la experiencia de mis directores en este tipo de trabajos, donde además de fijarse un objetivo desde un inicio, es importante adaptarse a los resultados parciales que se van obteniendo para así seguir el camino inicial, adaptarlo o trazar un camino alternativo.

Por otro lado, tratar un tema tan interesante como el de la detección de odio ha sido una gran motivación a lo largo del proyecto. Los meses durante los que se ha desarrollado (de marzo a julio de 2020) han sido realmente convulsos en el panorama político y social, por lo que la actividad en redes ha sido muy alta y ha requerido un trabajo de seguimiento constante. Este trabajo me ha mantenido motivado y concentrado en el proyecto durante estos meses.

Por último, cabe destacar que al inicio del proyecto mi conocimiento de aprendizaje automático y concretamente de Procesamiento del Lenguaje Natural era muy limitado. El desarrollo de este proyecto me ha ayudado a adquirir una visión general de estas disciplinas y el interés para seguir investigando en temas relacionados con el PLN y demás aplicaciones del aprendizaje automático.

6.2. Trabajo futuro

En la actualidad, las aplicaciones del aprendizaje automático en el área del Procesamiento de Lenguaje Natural son muy populares y abarcan una gran cantidad de ámbitos sociales. Gracias al análisis exhaustivo del panorama actual de la detección de odio mediante aprendizaje automático realizado en este proyecto, no solo se ha obtenido conocimiento de las distintas técnicas utilizadas para el Procesamiento del Lenguaje Natural, sino también de los acercamientos y sistemas *state-of-the-art* aplicables a muchos ámbitos más (como pueden ser la Visión por Computador).

Concretamente en el área del Procesamiento del Lenguaje Natural, el fin de este proyecto deja varias puertas abiertas a explorar.

- **Mantenerse al día** con las nuevas técnicas, *datasets* y sistemas desarrollados en

todo el mundo será importante en un campo tan popular como es el Procesamiento del Lenguaje Natural.

- El *dataset* construido en este TFG no ha cumplido las expectativas. El tiempo y el cariño que necesita la creación de un *dataset* de calidad queda fuera del alcance de un Trabajo de Fin de Grado. Sin embargo, con más tiempo para recoger mensajes de odio y aplicando los métodos aprendidos durante el proyecto y los que todavía son desconocidos para mí, la creación de un ***dataset* de discurso de odio en euskera** es un objetivo interesante a conseguir.

- Los sistemas desarrollados en el proyecto son ampliables no solo con un mejor uso de las técnicas utilizadas, sino también teniendo en cuenta que en los próximos meses y años se desarrollarán nuevas técnicas de aprendizaje profundo con los que podríamos **entrenar sistemas** con un mejor funcionamiento que los desarrollados en este proyecto. Además, se podría utilizar el *dataset* con discurso de odio en euskera para el entrenamiento de estos modelos.

- **Investigar la tendencia al odio que existe en Twitter dependiendo del idioma utilizado.** Una de las razones por las que el *dataset* nativo de tuits en euskera no contuviera carga de odio podría ser que el propio idioma no dé juego para ese tipo de mensajes. Esto se podría investigar analizando usuarios bilingües comparando sus tuits en castellano y euskera. De esta forma se podría demostrar si estos usuarios expresan odio en (por ejemplo) castellano, y no en euskera.

- Profundizar en el uso de *counter narratives* o **contranarrativas para lidiar con el discurso de odio online y la necesidad del aprendizaje automático** para ésta tarea. En un artículo de la Universidad de Trento [Tekiroglu et al., 2020] proponen la intervención textual (argumentos opuestos) en conversaciones online con discurso de odio como un buen método para contrarrestar los mensajes dañinos y evitar su propagación. La falta de corpus en este área (más aún en idiomas pequeños como el euskera) limita en gran manera el avance de estas investigaciones, por lo que sería interesante profundizar más en las aplicaciones de aprendizaje automático en este ámbito.

Anexos

A. ANEXO

Documento de Objetivos del Proyecto

A.1. Toma de decisiones

En primera instancia, los objetivos principales del proyecto eran los siguientes:

1. Entrenar modelos multilingües mediante deep learning para la detección de odio.
2. Crear un *dataset* con tuits en euskera con mensajes de odio.

Por supuesto, estos objetivos no son independientes uno del otro. Si bien los modelos se podrían entrenar con *datasets* de terceros como HatEval, nuestra intención era utilizar también nuestro *dataset* nativo en euskera para la creación de sistemas de detección de odio.

Una vez recogido el corpus en euskera, el siguiente paso fue establecer un límite inferior del rendimiento de los modelos a entrenar durante el proyecto. Este límite inferior o *baseline* lo marcan unos modelos simples para la detección de odio entrenados con la herramienta fastText.

Hasta este punto contamos con un corpus nativo de tuits en euskera y unos modelos simples para la detección de odio, por lo que el siguiente paso sería obtener modelos con mejores resultados que los creados en fastText. Para ello, se entrenan modelos multilingües partiendo del modelo pre-entrenado XLM-RoBERTa. Estos modelos *fine-tuneados* se evaluarían para probar su eficacia en la detección de odio en euskera. Dependiendo de los resultados obtenidos, el proyecto tomaría rumbos distintos:

- Si la predicción en euskera fuera buena con los modelos entrenados, habríamos conseguido el modelo multilingüe para la detección de odio en euskera que buscábamos desde un inicio (¡objetivo cumplido!) y podríamos utilizarlo para etiquetar nuestro corpus nativo en euskera. Una vez etiquetado, podríamos utilizarlo para entrenar un nuevo modelo y compararlo con los entrenados anteriormente.
- Sin embargo, si la predicción en euskera con los modelos entrenados no fuera buena, buscaríamos alternativas para conseguir modelos capaces de detectar odio en euskera. Una forma de hacer esto sería traduciendo el *dataset* HatEval en inglés o castellano al euskera (HatEval-EU) y entrenando un modelo con este corpus.

A.2. Planificación del proyecto

A.3. Paquetes de trabajo

A.3.1. Descripción de los paquetes

Aprendizaje (contextualización y estado del arte)

Este paquete engloba el trabajo realizado para la comprensión y afianzamiento de conocimientos relacionados con aprendizaje automático, profundo y sus aplicaciones en la detección de odio. Por tanto, se realiza un análisis del estado del arte en materia de PLN en detección de discurso de odio, así como de los sistemas y entornos a utilizar.

Datasets

Este paquete representa el trabajo realizado para la búsqueda, entendimiento y creación de *datasets* sobre discurso de odio, desde investigar cuáles son los más utilizados hasta la adaptación y creación de nuevos *datasets*.

Contiene tres subpaquetes:

- CREACIÓN DE CORPUS NATIVO EN EUSKERA: engloba el trabajo realizado para la creación del *dataset* de tuits en euskera, así como su posterior análisis y preprocesado.

- *DATASETS* HATEVAL (ADAPTACIÓN Y TRADUCCIÓN): engloba el trabajo realizado para la limpieza de los *datasets* de la competición HatEval de 2019, así como su adaptación a los formatos utilizados en este Trabajo de Fin de Grado. Incluye también el trabajo realizado en la traducción al euskera del *dataset* HatEval-ES para obtener HatEval-EU.

- ANÁLISIS DE LOS *DATASETS* CREADOS: engloba el trabajo realizado para el análisis de los *datasets* HatEval-ES, HatEval-EN, HatEval-EU y el *dataset* nativo.

Experimentación

Este paquete engloba el trabajo realizado para la creación de sistemas capaces de detectar odio en redes sociales, además de los sistemas para medir polaridad. Abarca desde el diseño de los experimentos hasta el propio entrenamiento de los modelos.

Contiene dos subpaquetes:

- CREACIÓN DE SISTEMAS PARA LA DETECCIÓN DE ODIO: engloba el trabajo realizado para el entrenamiento de modelos para la detección de odio. Este paquete se divide a su vez en dos partes: la primera destinada a los modelos entrenados con corpus HatEval-ES, HatEval-EN y la mezcla de ambos, y la segunda destinada a los modelos entrenados con el *dataset* traducido HatEval-EU y el intento con el corpus nativo de tuits euskera. Incluye también la configuración utilizada para el entrenamiento de los modelos, su evaluación y análisis.

- CREACIÓN DE SISTEMAS PARA MEDIR LA POLARIDAD: engloba el trabajo realizado para el entrenamiento de modelos para la medición de polaridad. Incluye también la configuración utilizada para el entrenamiento de los modelos, su evaluación y análisis.

Gestión del proyecto

Este paquete incluye todo el trabajo realizado para gestión del TFG, incluyendo la planificación, el Seguimiento y Control, las reuniones atendidas y el desarrollo de la memoria, el póster y presentación del Trabajo de Fin de Grado.

Por tanto, el paquete se divide en 5 partes esenciales:

- PLANIFICACIÓN: una parte esencial del paquete, pues en él se definen tanto los objetivos del proyecto como la metodología de trabajo a seguir para conseguir los entregables antes de las fechas límite.

- SEGUIMIENTO Y CONTROL: este subpaquete está presente desde el inicio del TFG hasta su fin. En él se plantean ideas tan importantes como el seguimiento del trabajo y la viabilidad del mismo (se mantiene a parte por el gran peso de las reuniones sobre la gestión del proyecto). Gracias al trabajo realizado en este paquete, nos aseguramos de que los objetivos se van cumpliendo y obtenemos una visión general del trabajo realizado y el que todavía queda por hacer.

- REUNIONES: este paquete podría pertenecer al paquete de Seguimiento y Control, pues su fin es el de asegurar que el trabajo avance adecuadamente. En las reuniones se establecen las pautas a seguir para el cumplimiento de los objetivos, así como la adaptación de los mismos en caso de que los objetivos originales no se fueran a cumplir.

- MEMORIA: se engloba el trabajo realizado para la creación de la memoria del Trabajo de Fin de Grado.

- PRESENTACIÓN Y PÓSTER: engloba el trabajo realizado para la creación del póster del TFG y el de las diapositivas para su presentación y defensa.

A.3.2. Diagrama EDT

La figura A.1 muestra el Diagrama de Descomposición del Trabajo o Estructura de Descomposición del Trabajo (EDT). Este esquema representa los paquetes de trabajo que componen el Trabajo de Fin de Grado. Está estructurado de la siguiente manera: una raíz, representando el TFG; un primer nivel, coloreado de un verde claro y representando los paquetes de trabajo principales del proyecto; un segundo nivel, coloreado de un verde intermedio y representando los subpaquetes que componen los paquetes del primer nivel; un tercer y último nivel, coloreado de un verde oscuro y representando los elementos básicos de los paquetes de segundo nivel. Todos los paquetes están explicados con mayor detalle en el apartado A.3.1.

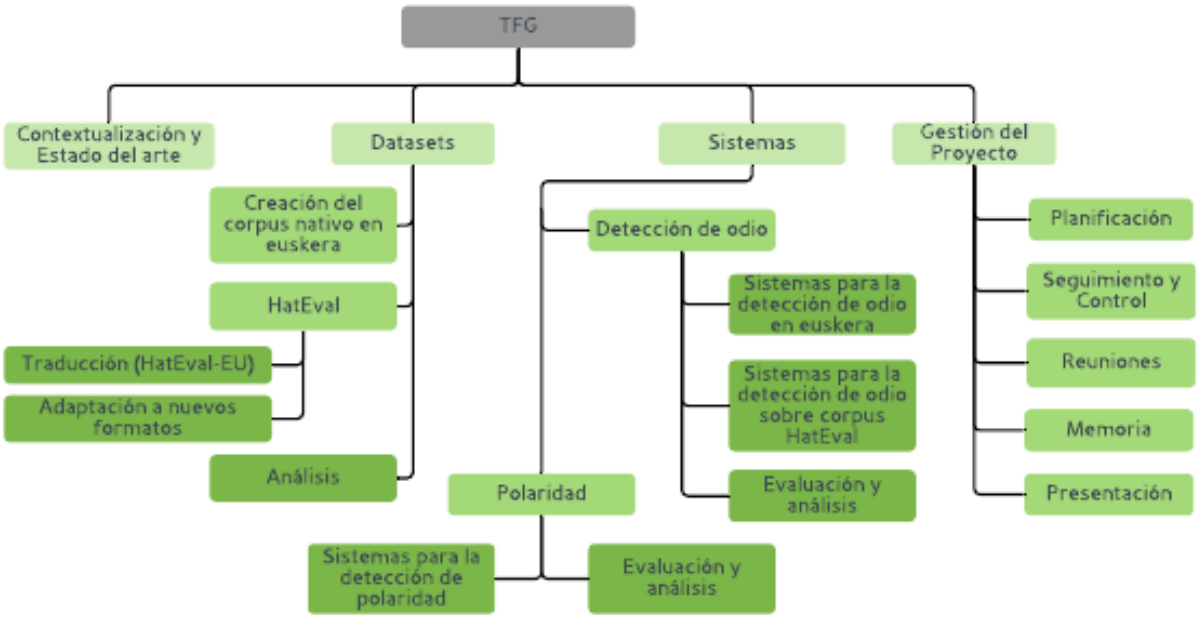

Figura A.1: Diagrama DTE.

A.3.3. Diagrama Gantt

Figura A.2: Diagrama Gantt.

A.3.4. Entregables, hitos y fechas límite

Los entregables del proyecto son los siguientes:

- Memoria
- Código, *datasets* y modelos entrenados[1]
- Póster

[1] https://github.com/Pafebla/tfg

- Presentación

La fecha límite para la entrega de la memoria, el póster del trabajo y el código, *datasets* y modelos es la misma: 6 de septiembre de 2020.

A.3.5. Dedicación y análisis de la desviación

La estimación de horas original del Trabajo de Fin de Grado fue de 346 horas.

En la planificación se le dio especial énfasis al aprendizaje y el estudio del estado del arte. Al tratar temas candentes como las arquitecturas neuronales Transformers donde a cada mes se publican nuevos artículos con distintas técnicas y acercamientos, estar al día en los mismos es de gran importancia.

Por otro lado, la creación de *datasets* constituye una parte importante de la estimación, pues es vital para la creación de los sistemas mediante aprendizaje automático que pretendíamos conseguir.

El paquete de experimentación no se queda atrás, pues aquí es donde se realiza el trabajo de entrenamiento de distintos modelos: desde los simples sistemas entrenados mediante fastText hasta los complejos Transformers.

Finalmente, la Gestión del Proyecto es la parte que más dedicación se le destina. Por un lado, porque es aquí donde nos aseguramos de que los objetivos se van cumpliendo y, por otro, porque aquí se encuentra el trabajo dedicado a la memoria.

PAQUETES DE TRABAJO		PREVISTAS	REALES	DESVIACIÓN
Aprendizaje (Estudio del estado del arte, sistemas a utilizar, entornos...)		45	50	5
Datasets		63	85	22
Creación de corpus nativo en euskera		40	55	15
Datasets HatEval: adaptación y traducción	Limpieza y adaptación a nuevos formatos	8	15	7
	Traducción (HatEval-EU)	5	5	0
Análisis de los datasets creados		10	10	0
Experimentación		93	121	28
Creación de sistemas para la detección de odio	Sistemas para la detección de odio sobre corpus HatEval	35	50	15
	Sistemas para la detección de odio sobre corpus traducido y nativo	25	40	15
	Evaluación y análisis	15	15	0
Creación de sistemas para medir la polaridad	Sistemas para la medición de polaridad	10	10	0
	Evaluación y análisis	8	6	-2
Gestión del proyecto		145	162,5	17,5
Planificación		10	10	0
Seguimiento y control		30	30	0
Reuniones		15	16,5	1,5
Memoria		75	85	10
Presentación y póster		15	21	6
		346	418,5	72,5

Figura A.3: Dedicación en horas y desviación.

Ánalisis de la desviación y el trabajo realizado

La dedicación total del Trabajo de Fin de Grado ha sido de 418,5 horas. Ha sufrido una desviación de 72,5 horas respecto a las 346 planeadas originalmente.

En general, ha sido un proyecto al que se le ha dedicado trabajo de manera constante. No solo por lo laborioso de la creación de un *dataset* como es el nativo de tuits en euskera que se ha recopilado, sino también por la dedicación necesaria para estar al día sobre los distintos acercamientos para la detección de discurso de odio o las arquitecturas neuronales Transformers.

En primer lugar, el paquete de Datasets ha sufrido una desviación de 22 horas. En un inicio se pensó que la recogida de tuits polémicos y con discurso de odio iba a ser más sencilla, pensando que los núcleos de odio (es decir, los grupos de personas con tendencia a expandir mensajes de odio) en Twitter no varían mucho. Sin embargo, con el inicio de la pandemia en marzo dio comienzo también a una actividad enorme entre los usuarios de Twitter: las tendencias cambiaban cada día y los mensajes de odio quedaron eclipsados por las tendencias sobre el COVID-19. Por tanto, el trabajo necesario para mantener el *dataset* al día fue más del esperado.

Por otro lado, la experimentación ha sido (después del paquete de Gestión del proyecto) el paquete que más trabajo ha requerido y ha sufrido 20 horas de desviación. Ésto es debido a que, en un inicio, el entrenamiento de sistemas inteligentes era un campo desconocido para mí y trabajar con arquitecturas modernas como los Transformers (donde los recursos en línea no son tan abundantes como los de otras arquitecturas) no ayudó. Sin embargo, según avanzaba el desarrollo del proyecto y me fui familiarizando con el entorno de HuggingFace's Transformers, el entrenamiento de modelos fue mucho más fluido.

Por todo ésto, la dedicación a la memoria se ha visto afectada, principalmente debido a necesidad de actualizar constantemente los datos del *dataset* en euskera y los resultados de la experimentación.

A.4. Riesgos y Prevención

En el desarrollo del proyecto se destacan unos riesgos y unas medidas a tomar para prevenir su aparición.

A.4.1. Riesgos

- **No conseguir un *dataset* en euskera con carga de odio**. Dos razones principales:
  1. Imposibilidad de obtener un *dataset* nativo de tuits en euskera. A la hora de obtener los tuits para el corpus, era posible la aparición de problemas respecto a la recolección: fallos en la API de Twitter, fallos de internet durante la recolección...
  2. El *dataset* de tuits nativos en euskera recogido no contiene suficiente carga de odio.
- **Pérdida parcial o total de archivos almacenados de manera local** (*datasets*, modelos, código...)
- **Pérdida parcial o total de la memoria desarrollada en Overleaf**[2].

A.4.2. Prevención

- **No conseguir un *dataset* en euskera con carga de odio**. Mediante la creación *dataset* traducido HatEval-EU nos aseguramos tener un *dataset* en euskera con carga de odio (en caso de imposibilidad de obtener *dataset* nativo de tuits en euskera o que la carga de odio del mismo sea insuficiente).
- **Pérdida parcial o total de archivos almacenados de manera local** (*datasets*, modelos, código...). Para evitar este escenario, mantenemos sincronizada nuestra copia del proyecto en la plataforma Google Drive[3].
- **Pérdida parcial o total de la memoria desarrollada en Overleaf**[4]. Para evitar este escenario, se crean copias locales al final de cada sesión de trabajo de memoria.

A.5. Seguimiento y Control

Se lleva a cabo un seguimiento constante del proyecto con el fin de asegurar el cumplimiento de los objetivos parciales y totales del proyecto.

[2] https://es.overleaf.com
[3] https://www.google.es/drive
[4] https://es.overleaf.com

Reuniones

Con el fin de mantener el proyecto organizado y establecer objetivos parciales, se mantienen comunicaciones semanales entre los directores y el estudiante (Tabla A.4). Estas reuniones se realizan a través del software Blackboard Collaborate[5], a excepción de las tres primeras, que se realizan de manera presencial en la Facultad de Informática de San Sebastián.

REUNIONES	
Fecha	Duración (horas)
13 febrero	1
20 febrero	1,75
10 marzo	0,75
17 marzo	0,75
25 marzo	0,75
8 abril	1
22 abril	0,5
29 abril	1
6 mayo	1
13 mayo	0,75
21 mayo	0,75
26 mayo	0,75
2 junio	1
8 junio	1
16 junio	0,75
23 junio	1
30 junio	0,5
7 julio	1
14 julio	0,5

Figura A.4: Reuniones llevadas a cabo en el proyecto.

Además de las reuniones, también se mantiene el contacto mediante correo electrónico.

Control de calidad

Se realiza un control de calidad constante sobre los *datasets* y sistemas desarrollados en el proyecto.

[5]https://www.blackboard.com

Datasets

En primer lugar, el control de calidad de los *datasets* de HatEval-ES y HatEval-EN se realiza investigando los análisis realizados por los concursantes de la competición HatEval en 2019. Gracias a este control se pudo, por ejemplo, detectar el problema del *dataset* HatEval-EN. En cuanto a HatEval-EU, el control se realizó comparando el *dataset* con su versión sin traducir, HatEval-ES.

Por otro lado, el control realizado sobre el corpus nativo de tuits en euskera fue realizado en el apartado 5.3.2 observando la proporción de etiquetas de odio.

Sistemas

El control de calidad de los sistemas se realizó evaluando cada uno de los modelos entrenados. Los resultados de las evaluaciones fueron comparados con los sistemas presentados en la competición HatEval, comprobando si los sistemas obtenían resultados a la altura de los mejores de la competición (ver Tabla 5.5).

A.6. Interesados

Los principales interesados en éste Trabajo de Fin de Grado son estudiantes, docentes e investigadores interesados en el Procesamiento de Lenguaje Natural y el aprendizaje automático.

Bibliografía

[Agerri et al., 2020] Agerri, R., Vicente, I. S., Campos, J. A., Barrena, A., Saralegi, X., Soroa, A., and Agirre, E. (2020). Give your text representation models some love: the case for basque.

[Basile et al., 2019] Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., and Sanguinetti, M. (2019). SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. pages 54–63.

[Bojanowski et al., 2016] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606.

[Cer et al., 2018] Cer, D., Yang, Y., yi Kong, S., Hua, N., Limtiaco, N. L. U., John, R. S., Constant, N., Guajardo-Céspedes, M., Yuan, S., Tar, C., hsuan Sung, Y., Strope, B., and Kurzweil, R. (2018). Universal sentence encoder. In submission.

[Davidson et al., 2017] Davidson, T., Warmsley, D., Macy, M. W., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. CoRR, abs/1703.04009.

[de Gibert et al., 2018] de Gibert, O., Perez, N., García-Pablos, A., and Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. pages 11–20.

[Indurthi et al., 2019] Indurthi, V., Syed, B., Shrivastava, M., Chakravartula, N., Gupta, M., and Varma, V. (2019). FERMI at SemEval-2019 task 5: Using sentence embeddings to identify hate speech against immigrants and women in twitter. pages 70–74.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. CoRR, abs/1907.11692.

[MacAvaney et al., 2019] MacAvaney, S., Yao, H.-R., Yang, E., Russell, K., Goharian, N., and Frieder, O. (2019). "hate speech detection: Challenges and solutions". PLoS ONE 14(8): e0221152.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. pages 1532–1543.

[Pérez and Luque, 2019] Pérez, J. M. and Luque, F. M. (2019). Atalaya at SemEval 2019 task 5: Robust embeddings for tweet classification. pages 64–69.

[Sherstinsky, 2018] Sherstinsky, A. (2018). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. CoRR, abs/1808.03314.

[Stappen et al., 2020] Stappen, L., Brunn, F., and Schuller, B. (2020). Cross-lingual zero- and few-shot hate speech detection utilising frozen transformer language models and axel.

[Tekiroglu et al., 2020] Tekiroglu, S. S., Chung, Y.-L., and Guerini, M. (2020). Generating counter narratives against online hate speech: Data and strategies.

[Tiedemann and Thottingal, 2020] Tiedemann, J. and Thottingal, S. (2020). OPUS-MT — Building open translation services for the World.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. CoRR, abs/1706.03762.

[Wang et al., 2018] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. CoRR, abs/1804.07461.

[Waseem, 2016] Waseem, Z. (2016). Are you a racist or am I seeing things? annotator influence on hate speech detection on twitter. pages 138–142.

[Waseem and Hovy, 2016] Waseem, Z. and Hovy, D. (2016). Hateful symbols or hateful people? predictive features for hate speech detection on twitter. pages 88–93.

[Wolf et al., 2019] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. ArXiv, abs/1910.03771.

[Zampieri et al., 2020] Zampieri, M., Nakov, P., Rosenthal, S., Atanasova, P., Karadzhov, G., Mubarak, H., Derczynski, L., Pitenis, Z., and Çağrı Çöltekin (2020). Semeval-2020 task 12: Multilingual offensive language identification in social media (offenseval 2020).

[Zhang and Luo, 2018] Zhang, Z. and Luo, L. (2018). Hate speech detection: A solved problem? the challenging case of long tail on twitter. CoRR, abs/1803.03662.