Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

CLASIFICACIÓN AUTOMÁTICA DEL TIPO DE DELITOS DE ODIO EN DENUNCIAS POLICIALES

Alumno: Juan Bautista Muñoz Ruiz

Tutor: Prof. D.ª Salud María Jiménez Zafra
Prof. D.ª María Teresa Martín Valdivia

Dpto: Informática

Septiembre, 2023

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Dª Salud María Jiménez Zafra y Dª María Teresa Martín Valdivia, tutores del Proyecto Fin de Carrera titulado: Clasificación automática del tipo de delitos de odio en denuncias policiales, que presenta Juan Bautista Muñoz Ruiz, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2023

El alumno:

MUÑOZ RUIZ JUAN BAUTISTA - 26516720C
Firmado digitalmente por MUÑOZ RUIZ JUAN BAUTISTA - 26516720C
Fecha: 2023.09.07 13:20:43 +02'00'

Juan Bautista Muñoz Ruiz

Los tutores:

JIMENEZ ZAFRA SALUD MARIA - 77370897R
Firmado digitalmente por JIMENEZ ZAFRA SALUD MARIA - 77370897R
Fecha: 2023.09.07 13:38:37 +02'00'

Salud María Jiménez Zafra

MARTIN VALDIVIA MARIA TERESA - 26006595N
Firmado digitalmente por MARTIN VALDIVIA MARIA TERESA - 26006595N
Fecha: 2023.09.07 16:37:16 +02'00'

María Teresa Martín Valdivia

ÍNDICE

Índice de imágenes

Índice de tablas:

Índice de ecuaciones:

1. INTRODUCCIÓN AL PROYECTO

1.1. Introducción

En la última década los delitos de Odio han sufrido un crecimiento prácticamente exponencial en la sociedad. Tal y como se recoge en el *"Informe sobre la evolución de los delitos de Odio*" publicado en el año 2021 se registró un incremento en torno al 24% con respecto al año 2019 en cuanto a cardinalidad de los delitos de racismo en nuestro país (Ministerio de Interior, 2022). Además, en dicha fuente se informa de un crecimiento en un 30% en los delitos hacia el género y preferencias sexuales de las personas.

Como consecuencia de esta tendencia a la alza, así como la contínua digitalización acelerada de las instituciones y servicios tras la pandemia del 2020, es de principal interés aplicar los conocimientos y diferentes tecnologías computacionales en este ámbito.

El proyecto a desarrollar consiste en un sistema clasificador para delitos de Odio entrenado con un corpus de denuncias cedido por el Ministerio del Interior.

1.2. Motivación

El estímulo principal a la hora de elegir este proyecto para la realización del Trabajo de Fin de Grado fue la posibilidad de seguir aprendiendo en campos que siempre me han despertado la curiosidad. Campos que están en la boca de todos a día de hoy como la Inteligencia Artificial, Procesamiento del Lenguaje Natural, Sistemas de Recuperación de Información, Machine Learning y Ciencia de Datos.

La posibilidad de automatizar una tarea tan ardua como la clasificación de denuncias policiales gracias a los diferentes modelos y tecnologías existentes resulta bastante atractiva, tanto a nivel personal como a nivel de generación de un producto de alto valor añadido y de utilidad para las instituciones.

Inicialmente se contó con un conjunto de atestados policiales proporcionado por el Ministerio del Interior dividido en dos categorías. Con este Trabajo de Fin de Grado se pretende desarrollar un sistema de clasificación automática de denuncias policiales que categorice la tipología del atestado de este corpus como delito de Odio o delito de No Odio.

Para la resolución del problema se ha desarrollado un proyecto que dispone de una aplicación web sencilla para la visualización de resultados y un sistema para el

entrenamiento de modelos, procesado de los atestados policiales y validación de los resultados.

1.3. Objetivos

- Realizar un estudio de los diferentes modelos de clasificación, técnicas de Procesamiento del Lenguaje Natural y Aprendizaje Automático aplicables.
- Diseñar y desarrollar un sistema clasificador automático de delitos de odio.
- Diseñar e implementar una aplicación de visualización de resultados intuitiva y sencilla.
- Redactar la memoria del proyecto con toda la información acerca del trabajo desarrollado.
- Redacción del manual de instalación y usuario.

1.4. Planificación de hitos y tareas

1.4.1 Metodología

Para el desarrollo del proyecto se ha utilizado una metodología ágil, es decir, un desarrollo iterativo en el que prima la implementación y creación de partes o incrementos del proyecto completamente funcionales de valor. Además, en cada nueva fase de trabajo del proyecto se han ido revisando y mejorando las secciones anteriormente implementadas conforme se iban aprendiendo nuevas tecnologías y formas más óptimas de codificación.

1.4.2 Planificación

Cabe destacar que el desarrollo de este Trabajo de Fin de Grado ha sido solapado con el cursado de la asignatura Procesamiento del Lenguaje Natural en el estudio de algunas tecnologías.

A continuación, se dispone de una tabla a modo de resumen de la planificación seguida, así como un diagrama de Gantt para una visualización más orientativa.

ACTIVIDAD	**DURACIÓN**
Introducción a Python.	1 semanas
Estudio de los Sistemas Clasificadores.	1 semanas
Estudio de la tecnología Transformers y modelos BERT.	2 semanas
Estudio del Procesamiento de texto (Tokenización, lematización, stop words…).	2 semanas
Implementación del Primer Modelo BERT funcional.	2 semanas

Implementación de un Modelo BERT clasificador de denuncias inicial.	3 semanas
Estudio de algunos de los Diferentes modelos existentes (k-Means, SVM, Naive Bayes, Regresión Logística, Árboles de Decisión y Redes Neuronales).	2 semanas
Estudio e implementación de la validación cruzada k-fold.	1 semanas
Estudio de las métricas de evaluación (Precision, Recall, F1-score…) e implementación en el sistema.	3 semanas
Estudio del Framework Web Flask.	3 semanas
Desarrollo de la aplicación Web con Flask.	4 semanas
Implementación de otros modelos alternativos.	2 semanas
Estudio y evaluación del rendimiento de los diferentes modelos implementados.	3 semanas
Desarrollo de la memoria.	4 semanas
Desarrollo del manual de instalación.	1 semanas
Revisión de código y entrenamiento de los modelos	4 semanas
Corrección y revisión de la memoria del proyecto	4 semanas

Tabla 1: Planificación del proyecto

Ilustración 1: Diagrama de Gantt Parte 1 (Elaboración propia)

Ilustración 2: Diagrama de Gantt Parte 2 (Elaboración propia)

1.5. Presupuesto

En este apartado se realizará una estimación de costes para la planificación anterior. Estos cálculos serán aplicados sobre una duración total de 300 horas tal y como se indica en la guía docente del TFG.

1.5.1. Costes en Software

El proyecto será desarrollado con varios softwares de soporte gratuitos junto con PyCharm. Este último es el entorno de desarrollo de la empresa JetBrains, aunque ha sido realizado con una licencia educativa, supondremos la compra de una licencia anual de este software. El coste de esta es de 249,00 €.

Software	Coste de la licencia
PyCharm	**249,00 €**
Canva	**0 €**
Entorno de Google Drive	**0 €**
Diagrams.net	**0 €**
Github	**0 €**
Total	**249,00 €**

Tabla 2: Costes en software

1.5.2. Costes en Hardware

En este apartado se tienen en cuenta los costes del ordenador de trabajo, unos cascos para las reuniones telemáticas y el consumo de luz, tanto del ordenador como del router de internet.

Cabe destacar que los resultados obtenidos son estimaciones y pueden estar sujetos a la fluctuación del precio del kW/hora.

Hardware	Coste
Portatil Asus Intel i7-3630QM CPU 2.40GHz 8GB Ram	**799 €**
Tarifa de Internet	70 €/mes x 7 meses = **490 €**
Consumo del portátil (90 W)	Consumo diario = 90 *W / 1000 kW* = 0,9 *kW/hora* Coste del kW/h = 0,18509 €/h Horas diarias = (300 horas / 7 meses / 24 *horas/día*) = 1,7857 horas/día Gasto diario = 0,9 kW/h x 0,18509 €/h x 1,7857 horas/día = 0,02974 €/día Días en 7 meses= 212 días Consumo de luz = 212 días x 0,02974 €/día = **6,31 €**
Consumo del router (4,5 W)	Consumo diario = 4,5 *W / 1000 kW* = 0,0045 *kW/hora* Coste del kW/h = 0,18509 €/h Horas diarias = (300 horas / 7 meses / 24 *horas/día*) = 1,7857 horas/día Gasto diario = 0,0045 kW/h x 0,18509 €/h x 1,7857 horas/día = 0,001487 €/día Días en 7 meses= 212 días Consumo de luz = 212 días x 0,001487 €/día = **0,31 €**
Cascos para vídeo llamada	**49,99 €**
Total	**1.345,61 €**

Tabla 3: Costes en hardware

1.5.3. Costes en Recursos Humanos

Para esta sección se tendrá en cuenta el salario medio de un programador analista de datos.

Recursos Humanos	Coste
Salario de programador analista de datos	1974 €/mes x 7 meses = **17.766 €**
Total	**17.766 €**

Tabla 4: Costes en recursos humanos

1.5.4. Costes no previstos

Los gastos no previstos se calcularán con una estimación en torno al diez por ciento sobre el total de cara a tener un presupuesto adaptable a posibles fallos o imprevistos durante el desarrollo.

Costes no previstos	Coste
Costes no previstos	Costes no previstos = 0,1 x (software + hardware + recursos humanos) Costes no previstos = (249 € **+**1974 345,61 € 17,766 €) x 0,10**= 1.776 €**
Total	**1.776 €**

Tabla 5: Costes no previstos

1.5.5. Costes totales

Para los gastos totales procederemos a sumar todos los costes anteriormente calculados. Tal y como se vislumbra en el siguiente desglose, los costes totales para el desarrollo del proyecto ascienden a 21.136,61 €.

Presupuesto	Coste
Costes en software	249 **€**
Costes en hardware	1.345,61 €
Costes en recursos humanos	17.766 €
Costes no previstos	1.776 €
Total	**21.136,61 €**

Tabla 6: Costes totales

2. ESTUDIO DE LAS TECNOLOGÍAS EXISTENTES Y NECESARIAS: MODELOS CLASIFICADORES

Como estudio previo a la resolución del problema de la clasificación de denuncias policiales, se han realizado una serie de investigaciones en diversas

temáticas para entender, comprender y valorar las diferentes tecnologías y metodologías dentro del estado de la cuestión de los modelos clasificadores.

2.1. Sistemas de Clasificación

Para comenzar el desarrollo de este proyecto era de vital importancia saber qué era un sistema clasificador, así como sus principales características y marco tecnológico en el que encuadrarlo.

Desde la perspectiva del campo del Procesamiento del Lenguaje Natural se disponen de los siguientes tipos de aplicaciones o sistemas (Ureña, s.fa):

2.1.1. Sistemas según su fin

Según esta clasificación los sistemas se estructuran en función de la consideración como procesos finales o no finales.

2.1.1.1. Sistemas considerados finales

Son sistemas cuya tarea realizada es percibida como objetivo definitivo en el proceso computacional seguido. En este tipo de aplicaciones podemos encontrar: Sistemas de Clasificación, Sistemas de Traducción, etc… (Ureña, s.fa)

2.1.1.2. Sistemas considerados no finales

Se trata de sistemas cuya tarea realizada no es percibida como objetivo definitivo en el proceso computacional seguido, es decir, son un paso intermedio en la consecución del objetivo por parte de otra aplicación final. Por ejemplo, la conversión de palabras a tokens y la extracción de entidades (Ureña, s.fa).

2.1.2. Sistemas según su orientación

En esa agrupación se reúnen los sistemas en base al ámbito informático en el que se aplican (Ureña, s.fa):

2.1.2.1. Sistemas con campo de aplicación en el procesamiento textual

Son aplicaciones fundadas en el procesamiento de texto con el objetivo de que los computadores sean capaces de crear y comprender material escrito con mayor precisión de cara a automatizar procesos (Ureña, s.fa). En este grupo podemos encontrar los Sistemas Clasificadores, Sistemas Traductores, Sistemas Generadores de Resúmenes, etc… (Ureña, s.fa).

2.1.2.2. Sistemas con campo de aplicación en la comunicación entre el hombre y el computador

Estos sistemas están basados en el estudio del ser humano y el lenguaje natural de cara a mejorar y facilitar la comunicación entre las personas y los ordenadores (Ureña, s.fa).

Para este proyecto se desarrollará un sistema clasificador de atestados o denuncias policiales. Como podemos inferir del anterior análisis de tipología de las

aplicaciones dentro del Procesamiento del Lenguaje Natural, los Sistemas Clasificadores de Texto son **sistemas finales y fundamentados en el procesamiento textual**, es decir, son aplicaciones cuya tarea realizada es considerada definitiva y que se centran en el procesamiento de las palabras de cara a automatizar alguna tarea, en este caso la clasificación.

2.1.3. Sistemas según el etiquetado

En cuanto a la definición de un Sistema Clasificador. Clasificar consiste en asignar una o varias etiquetas o categorías por cada documento.

En esta división de los sistemas clasificadores encontramos tres grandes grupos. Tal y como se detalla a continuación, un sistema pertenece a una categoría u otra en función del número de clases de entre las que se puede clasificar y del número con las que finalmente clasifica. Estos tipos son los sistemas binarios, multi-etiqueta y multi-clase (Paul, 2021):

2.1.3.1. Sistemas Binarios

Son sistemas en los que en la clasificación se dispone de dos etiquetas o categorías pero sólo se asigna una al documento (Paul, 2021). Por ejemplo, una clasificación en la que un mensaje de texto puede ser Ofensivo o No Ofensivo.

2.1.3.2. Sistemas Multi-Etiqueta

En estos sistemas disponemos de varias etiquetas o categorías pero solo optamos a clasificar el documento con una (Paul, 2021). Por ejemplo, un texto que puede ser Deportivo, de Moda o Económico pero únicamente es clasificado como Económico.

2.1.3.3. Sistemas Multi-Clase

Para la clasificación el sistema también puede elegir de entre varias categorías, pero a diferencia del anterior, el documento o texto puede ser etiquetado con varías (Paul, 2021). Siguiendo el ejemplo de categorías anterior, se podría clasificar un texto como Económico y Deportivo.

En este caso, el sistema clasificador a desarrollar será un **sistema binario**. Esta decisión de diseño está fundamentada en la propia estructura del corpus proporcionado por el Ministerio de Interior. Este conjunto de textos cuenta con dos principales clases de etiquetado, Odio y No Odio, de entre las cuales cada denuncia sólo pertenece a una de ellas.

Por otro lado, siguiendo con el estudio de los posibles modelos aplicables a la resolución del problema, encontramos tres categorías de sistemas clasificadores según los modelos utilizados (Ureña, s.fa):

2.1.4. Sistemas según el modelo de clasificación

Los sistemas serán considerados de un tipo u otro según el entrenamiento, programación y estructura algorítmica con el que han sido implementados.

2.1.4.1. Sistemas Basados en Reglas

Son sistemas en los que se dispone de un árbol jerárquico de reglas preestablecidas por uno o varios expertos en los que la clasificación de un documento depende del cumplimiento de unas u otras. Suelen tener un mayor gasto en términos de implementación y de desarrollo. Sin embargo pero con resultados de alta eficacia (Ureña, s.fb).

2.1.4.2. Sistemas Basados en Machine Learning

Estos sistemas utilizan modelos de aprendizaje automático. Tienen la capacidad de volver a ser reusados y ampliados (Ureña, s.fb). Es decir, un mismo modelo puede ser entrenado con diferentes datasets o en este caso corpus de datos para realizar diferentes clasificaciones.

Algunos de los modelos más utilizados serán estudiados en este documento en las próximas líneas.

2.1.4.3. Sistemas Híbridos

En este tipo de sistemas se suelen combinar los dos anteriores quedándo con lo mejor de ambas metodologías.

Para el problema en concreto de la clasificación de atestados policiales nos centraremos en los **sistemas basados en Machine Learning**. Se ha tomado esta decisión, ya que no contamos con conocimiento experto para la elaboración de un sistema basado en reglas.

2.2. Estructura de un Sistema Clasificador de texto

La estructura general de un sistema clasificador, en base a lo aprendido en las prácticas de la asignatura Procesamiento del Lenguaje Natural, es la que podemos observar en el siguiente esquema (Cámara, 2023). Para otros tipos de clasificadores se seguiría la misma estructura pero cambiando la tipología de corpus.

Por un lado, encontramos un corpus con el conjunto de textos, el cual se divide en un conjunto de validación y entrenamiento junto con un texto a clasificar extraído de este corpus (Ureña, s.fb).

Por otro lado, encontramos un modelo clasificador que aprende del conjunto de entrenamiento y un sistema de evaluación que calcula unas métricas con base en los resultados del conjunto de evaluación. Finalmente, este texto extraído del corpus recibido como entrada por el clasificador generando como resultado la predicción de una etiqueta (Cámara, 2023).

Ilustración 3: Estructura de un Sistema Clasificador (Elaboración propia)

2.2.1. Corpus

Cuando hablamos de corpus nos referimos a una colección de documentos de texto. Es utilizada en el campo del Procesamiento del Lenguaje Natural para el entrenamiento y la validación de los sistemas. Pueden estar en uno o varios idiomas, en formato escrito o hablado (Ureña, s.fc).

Una vez establecido un corpus lo dividiremos en dos conjuntos. Uno de entrenamiento y otro de validación.

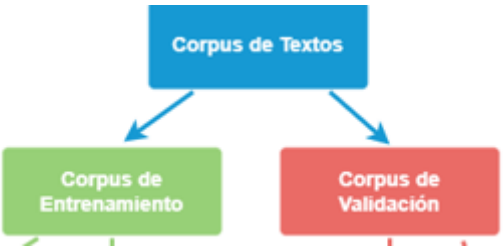

Ilustración 4: Corpus de un Sistema Clasificador (Elaboración propia)

El corpus de entrenamiento será utilizado para alimentar a nuestro modelo mientras que el corpus de validación será utilizado para calcular las métricas de nuestro modelo. Es interesante realizar esta división de una forma equilibrada obteniendo un reparto equitativo de etiquetas de cara a obtener un entrenamiento balanceado del modelo (Cámara, 2023). Además, el texto de ambos conjuntos debe ser tratado de igual manera en cuanto a procesamiento e inferencia del vector de características.

2.2.2. Modelo Clasificador

El clasificador es el epicentro del sistema y es entrenado con el corpus de entrenamiento. Recibe como entrada los vectores de características de los textos procesados y sus etiquetas, si es que las tiene, sobre los que aplicará alguno de los procedimientos matemáticos explicados en el punto 2.3. Tras este entrenamiento el sistema será capaz de realizar predicciones de a qué etiqueta o clase pertenece un texto distinto a los que ha recibido como entrenamiento (Cámara, 2023).

2.2.3. Sistema de Validación

Es necesario someter a evaluación un modelo clasificador para medir su grado de éxito en la clasificación o cualquier otra tarea para la que esté desarrollado el modelo (Cámara, 2023).

Ilustración 5: Validación en Sistema Clasificador (Elaboración propia)

Recibe como entrada un vector de etiquetas predichas por el modelo para unos textos y un vector o conjunto de etiquetas reales de esos mismos textos. De la

comparación de esos dos conjuntos de etiquetas surgen las métricas explicadas en el punto 2.6.

2.2.4. Texto a Clasificar

El objetivo principal de un modelo de categorización es la clasificación. Se trata de la etapa final, tras el entrenamiento y la validación. Es de gran importancia que este texto sea sometido al mismo pre-procesamiento y generación de vectores de características que ha sufrido el corpus de entrenamiento para que la predicción funcione correctamente (Cámara, 2023).

2.3. Técnicas para el preprocesamiento de un texto

2.3.1. Tokenización

La tokenización de un texto consiste en la selección de los componentes que se procesarán. Estos componentes reciben el nombre de tokens. Se recibe como entrada una cadena texto y obtenemos una lista de subcadenas en la salida (Ureña, s.fd).

Podemos encontrar o implementar gran variedad de tokenizadores según el tipo de división a realizar sobre el texto de entrada. Esta división puede ir desde frases, palabras hasta una división en carácteres. Por otro lado, también podemos establecer diferencias en cuanto la integración en el conjunto final de tokens de los números y de los signos de puntuación (Ureña, s.fd).

Para una clasificación de texto, en este caso una categorización de denuncias policiales, en términos de diseño la tokenización ideal es una tokenización en palabras que no tenga en cuenta a los signos de puntuación. En palabras, ya que esta tokenización tendría mayor relevancia a la hora de entrenar nuestro modelo con una serie de elementos instanciables sin que sea excesiva ni escasa. Por otro lado, sin signos de puntuación porque para tareas de clasificación estos carecen de importancia, a diferencia de otro tipo de sistemas como generadores de resúmenes o extractores de entidades en los que sí tendrían mayor impacto.

2.3.2. Lematización y Stemming

Es de especial interés la reducción de los tokens a una instancia común. De esta manera, obtenemos una representación única dotando así a nuestro sistema de mayor rendimiento Un modelo de clasificación perdería eficacia si una palabra se encontrara en diferentes formatos, ya que en el entrenamiento en número de representaciones de una palabra estaría dividido en tantos grupos como formatos tenga la palabra en lugar de uno único común.

En la lematización se reduce a una palabra base preestablecida llamada lema. Para nombres y adjetivos estos lemas suelen ser convertidos a masculino singular, mientras que para verbos quedaría reducida al infinitivo (Ureña,s.fd). Un ejemplo:

- Cochera -> Coche
- Librería -> Libro

En el Stemming se reduce el token a la raíz. Por lo que el sistema lematizador se queda con la parte de la palabra que no es ni sufijo ni es prefijo (Ureña,s.fd):

- Cochera -> Coch
- Librería -> Libr

2.3.3. Eliminación de Stop Words

Este paso consiste en eliminar de la lista de tokens aquellos que no aportan información, ya que aparecen con gran frecuencia en los textos o incluso demasiado poco (Yoseo, 2017). Todo ello con el objetivo final de aumentar el rendimiento del modelo mediante la reducción del número de tokens.

Se podría cargar un recurso lingüístico para cada idioma con el listado de palabras vacías o incluso crear un propio listado con las palabras que carecen de utilidad para nuestro modelo en un ámbito específico (Ureña,s.fd).

2.3.4. Vectores de Características o Word-Embedding

Los modelos de entrenamiento están preparados para trabajar en formato numérico, por lo que es de vital importancia obtener una representación en este formato (Gautam, 2021). Por cada documento o texto se genera un vector. Encontramos los word embeddings de frecuencia y los word embeddings de contexto (Gautam, 2021).

2.3.4.1. Word-Embedding de frecuencia

Esta extracción de características está fundamentada en el número de apariciones de las palabras de cada frase (Gautam, 2021).

2.3.4.1.1. Bag-of-words:

Se reúnen todas las palabras tokenizadas que son distintas en un listado. Por cada documento se generará un vector con la misma longitud que este listado. Cada posición se corresponderá con cada palabra del texto. En esta posición se guardará el conteo de cada palabra. También puede ser conocido como CountVectorizer (Murzone, 2022).

Frase	Primer	segundo	ejemplo	de	frase
Primer ejemplo de frase	1	0	1	1	1
Segundo ejemplo de frase	0	1	1	1	1
ejemplo ejemplo ejemplo	0	0	3	0	0

Tabla 7: Ejemplo de Bag-of-words

Cabe destacar que con este vector de características se puede tener en cuenta el orden o colocación contando la presencia de conjuntos de dos o varias palabras (Murzone, 2022).

2.3.4.1.2. TF-IDF:

Este enfoque nace para solucionar la problemática del gasto computacional respecto a la anterior metodología con conjuntos de gran tamaño (Murzone, 2022). Trabaja teniendo en cuenta dos variables realizando un producto entre ellas. Estas variables son la frecuencia de término (Term Frequency, TF) y la frecuencia inversa de término (Inverse Document Frequency, IDF) (Murzone, 2022).

$$TFIDF = TF * IDF$$

(Fórmula 1: Tf-idf)

La Frecuencia de Término, en una versión simple, puede ser representada mediante el número de instancias de la palabra dividido entre el número de palabras totales del texto (Murzone, 2022).

$$TF = \frac{Número\ de\ instancias\ de\ la\ palabra\ en\ el\ texto}{Número\ de\ palabras\ totales\ del\ texto}$$

(Fórmula 2: Frecuencia de término)

La Frecuencia de Término Inversa es calculada mediante el logaritmo neperiano de la división efectuada entre el número total de textos y el número de textos en los que aparece la palabra buscada (Murzone, 2022).

$$IDF = \ln\left(\frac{Número\ de\ textos}{Número\ de\ textos\ en\ los\ que\ aparece\ la\ palabra}\right)$$

(Fórmula 3: Frecuencia de término inversa)

2.3.4.1.3. Hash-Vectorizer:

El funcionamiento es el mismo que el Bag-of-words. Sin embargo, este no almacena todas las palabras en una lista a modo de índice. A cada palabra se le realiza una transformación hash que convierte la palabra en un número. Posteriormente este número es usado como índice de la tabla (Ganesan, 2020).

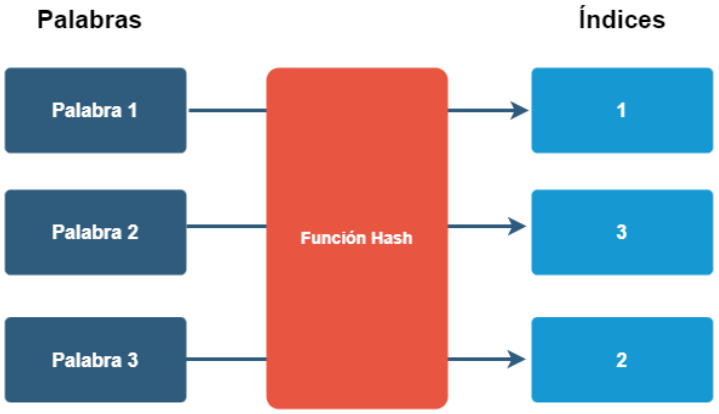

Ilustración 6: Vectorizador Hash (Elaboración propia)

2.3.4.2. Word-Embedding de contexto

A diferencia de los anteriores estos words embedding tienen en cuenta el entorno de la palabra y otros factores como el significado. Son capaces de condensar toda esta información en un vector de números reales (Gautam, 2021).

Para dos palabras relativamente parecidas obtendremos vectores numéricos cercanos (Gautam, 2021). A continuación, se presenta un ejemplo de word embedding imaginario siguiendo cuatro dimensiones ficticias:

Palabra	Velocidad	Seguridad	Cotidianeidad	Número Ruedas
Coche	0,7	0,7	0,8	1
Moto	0,75	0,6	0,4	0,5
F1	1	0,9	0,01	1
triciclo	0,01	0,1	0,1	0,75
Bicicleta	0,2	0,3	0,2	0,5

Tabla 8: Word embedding de contexto (Elaboración propia)

Como podemos observar cada palabra tiene un vector único según su contexto. Cuanto más parecidas sean dos palabras, más cercano a 1 será el producto vectorial de su vector de características. Un ejemplo de este tipo es WordtoVec.

2.3.4.2.1. Word2Vec

Word2Vec se trata de un modelo para la generación de vectores de características que trabajaba usando una red neuronal de varias capas. Distinguimos dos algoritmos en este vectorizador de características. Continuous Bag of Words y Skip Gram. El primero se fundamenta en la predicción de palabras

recibiendo como entrada un contexto y el segundo en la predicción de contextos dada como entrada una palabra (Monica, 2022).

2.4. Modelos de Clasificación

Como vimos en el punto 2.2, un modelo en el ámbito del aprendizaje automático es un sistema entrenado con un conjunto de datos que es capaz de generar una salida con una predicción en base a un input o entrada.

En el caso de un Sistema Clasificador de Texto, será aquel capaz de predecir a qué etiqueta pertenece un texto en función del corpus con el que ha sido entrenado. Encontramos los siguientes tipos de modelos:

2.4.1. No supervisados

Son aquellos en los que el conjunto de entrenamiento, o en este caso el corpus de entrenamiento, no tiene definidas etiquetas (Universidad Europea, 2022). Es decir, no se sabe la clase o tipo al que pertenece cada archivo. Son modelos que en función de coincidencias comunes crean unos conjuntos de palabras llamados Clusters (Universidad Europea, 2022). Para una clasificación de texto, un documento será de una categoría u otra según su cercanía con un cluster u otro (Cámara, 2023).

Uno de los modelos no supervisados más usados es el modelo k-Means. Este modelo reúne las palabras en unos *k* conjuntos llamados centroides. La posición de cada centroide es ajustada conforme se van insertando más elementos (Universidad de Oviedo, d.f).

En este problema en concreto, como contamos con un corpus etiquetado, no merece la pena implementar **un modelo no supervisado**, ya que perdemos toda la información que nos otorga contar con un corpus clasificado por un experto. Por lo que no se profundizará más en este tipo de modelos.

2.4.2. Supervisados

En este tipo de modelos sí se cuenta con un conjunto de elementos con etiquetas asignadas indicando su tipo (Universidad Europea, 2022). El modelo realizará la predicción de la etiqueta en base a las etiquetas de los textos con los que ha sido entrenado.

A continuación, se describirán algunos de los principales modelos supervisados.

Por un lado, encontramos modelos con los que se han realizado pruebas como **SVM, Naive Bayes y las diferentes variantes de BERT**. De entre ellos, SVM y Naive Bayes fueron elegidos para poder comparar resultados con otros proyectos de la asignatura Procesamiento del Lenguaje Natural, mientras que BERT fue elegido para entender y aprender a usar dicha tecnología.

Por otro lado, encontramos modelos que no han sido sujetos a prueba en la implementación, como K-NN, Regresión Lineal, Redes Neuronales y Transformers. Estos también serán descritos al ser de especial interés para estudiar y entender el funcionamiento de evoluciones posteriores como los modelos BERT.

2.4.2.1. Support Machine Vector (SVM)

Esta clasificación está basada en la búsqueda del plano que separe los elementos de dos tipos de la forma más eficiente. Es decir, se construye la línea formada por los puntos que se encuentren a la máxima distancia de los elementos de ambos tipos (Rodriguez, 2021).

En caso de contar con varias categorías, se generará un plano recombinado con los generados por cada par de ellas. La clasificación o asignación de una etiqueta u otra será realizada en función de la posición con respecto a ese plano (Rodriguez, 2021).

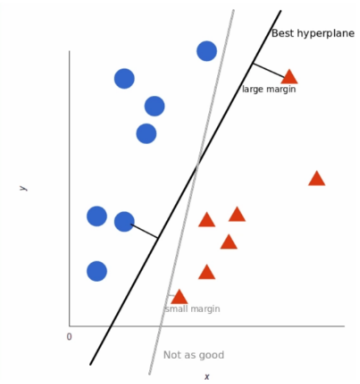

Ilustración 7: Support Machine Vector (Rodriguez, 2021)

2.4.2.2. Naive Bayes

Este modelo trabaja fundado en el teorema de Bayes. Este teorema calcula la probabilidad de que ocurran dos eventos en función de la probabilidad de que ocurran estos por separado (Stecanella, 2017).

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

(Fórmula 4: Probabilidad Condicional)

De cara al cálculo de la probabilidad de que una palabra pertenezca a cierta etiqueta. En primer lugar, se multiplicará la probabilidad de que en un texto de esa etiqueta aparezca la palabra por la probabilidad de que un texto pertenezca a esa etiqueta. Posteriormente, el resultado obtenido se dividirá entre la probabilidad de que aparezca esa palabra en un texto:

$$P(palabra|etiqueta) = \frac{P(etiqueta|palabra) \cdot P(etiqueta)}{P(palabra)}$$

(Fórmula 5: Ejemplo de Probabilidad Condicional)

Para clasificar una frase o un texto completo basta con efectuar un producto general de las probabilidades de que cada palabra de este pertenezca a cada etiqueta (Stecanella, 2017). La etiqueta que obtenga mayor probabilidad será la que el modelo establezca como predicción:

$$P\Big(texto|etiqueta\ 1\Big) = P\Big(palabra\ 1|etiqueta\ 1\Big) \times P\Big(palabra\ 2|etiqueta\ 1\Big) \times \ldots\ldots\ldots \times P\Big(palabra\ n|etiqueta\ 1\Big)$$

$$P\Big(texto|etiqueta\ 2\Big) = P\Big(palabra\ 1|etiqueta\ 2\Big) \times P\Big(palabra\ 2|etiqueta\ 2\Big) \times \ldots\ldots\ldots \times P\Big(palabra\ n|etiqueta\ 2\Big)$$

(Fórmula 6: Cálculo de la predicción de una etiqueta)

2.4.2.3. K-Nearest Neighbors

El modelo de los *k* vecinos más cercanos utiliza la distancia euclídea para calcular esta distancia con respecto a cada vecino (Patel, 2019). Donde x1 e y1 son las coordenadas del punto 1, mientras que x2 e y2 son las coordenadas del punto 2.

$$Distancia\ euclídea\ (p1, p2) = \sqrt{\Big(x_2 - x_1\Big)^2 + \Big(y_2 - y_1\Big)^2}$$

(Fórmula 7: Distancia Euclídea)

A modo de inferencia, la adaptación de la fórmula de la distancia euclídea para un modelo clasificador haciendo uso de dos vectores de características v1 y v2 calculando la distancia entre cada posición quedaría así:

$$DE\ (v1, v2) = \sqrt{\Big(v2[0] - v1[0]\Big)^2 + \Big(v2[1] - v1[1]\Big)^2 + \ldots\ldots + \Big(v2[n] - v1[n]\Big)^2}$$

(Fórmula 8: Ejemplo Distancia Euclídea)

Posteriormente se escogen los *k* vecinos cuya distancia euclídea tiene menor tamaño, los más cercanos. La etiqueta que sea predominante en estos vecinos más cercanos será la elegida por el clasificador como la etiqueta a asignar en la predicción (Patel, 2019).

Vecino	Distancia Euclídea	Etiqueta
1	0.89	Etiqueta 1
2	1,67	Etiqueta 2
3	3,12	Etiqueta 1

Tabla 9: Clasificación KNN

Como la etiqueta mayoritaria en el cálculo de los 3 vecinos más cercanos en la etiqueta 1, el sistema clasificará la entrada recibida como etiqueta 1.

2.4.2.4. Regresión Lineal

El funcionamiento de este modelo está basado en la ecuación de la recta (Heras, 2020a).

$$y = a_0 + a_1 x$$

Donde:

x= Entrada.
y= Salida.
a0 = Punto donde la recta corta al eje y.
a1 = Pendiente de la recta.

(Fórmula 9: Ecuación de la recta)

Este modelo se encarga de encontrar la recta imaginaria que mejor encaje con el patrón lineal generado por los valores con coordenadas (x,y) conocidos. Con esta recta generada el modelo será capaz de realizar predicciones de valores desconocidos (Dot, 2017).

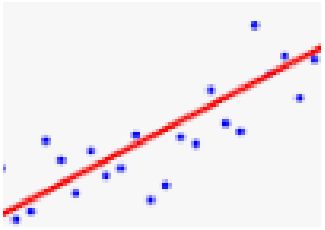

Ilustración 8: Regresión Lineal (Heras, 2022)

En caso de encontrarnos con varias rectas debido a la presencia de muchos más factores o variables de entrada independientes se generará un espacio vectorial mediante el conglomerado de las ecuaciones de cada recta (Dot, 2017).

2.4.2.5. Redes Neuronales

Las redes neuronales, por definición, engloban una serie de modelos que tratan de replicar el funcionamiento del aprendizaje y pensamiento humano. En la

estructura de una red neuronal encontramos una capa de entrada, una capa de salida y una o varias capas intermedias llamadas capas ocultas. Dentro de estas capas encontramos las neuronas o perceptrones.

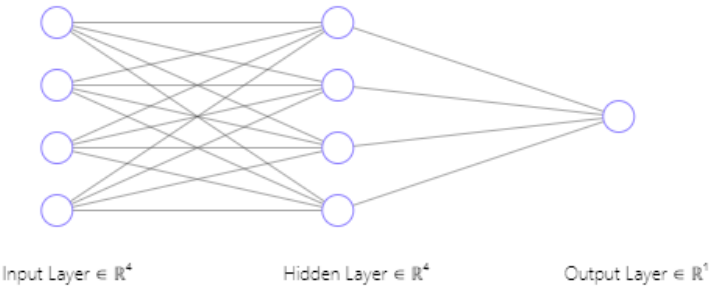

Ilustración 9: Red Neuronal (Elaboración propia)

Cada nodo o neurona que estén conectados entre sí tendrán asignado un peso. Este peso simboliza qué importancia tiene dicha neurona para la neurona con la que está conectada (Dot, 2018a).

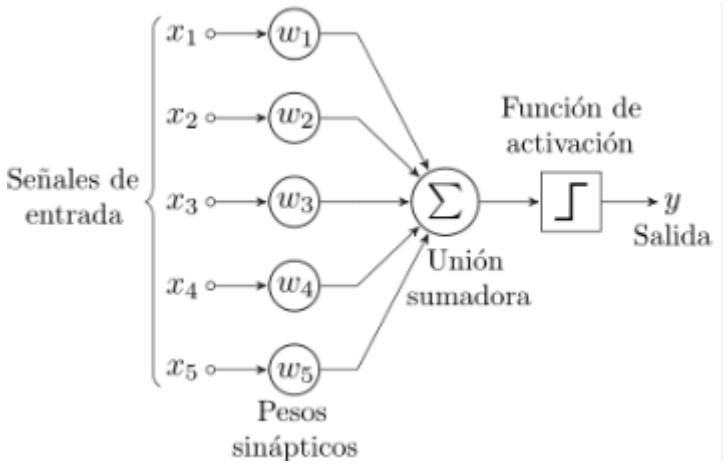

Ilustración 10: Neurona (Wikipedia, 2023a)

Por cada neurona de la capa oculta se calcula la sumatoria del producto del valor de las entradas con su respectivo peso (Wikipedia, 2023a). Como parámetro adicional, usaremos el bias. Al añadir este parámetro obtenemos una ecuación lineal de una recta, al igual que en la regresión lineal. Con este valor cumplirá la misma función que en esta, definir el punto de corte de la ecuación con el eje Y, permitiendo el desplazamiento de la recta a lo largo del eje (Dot, 2018a).

$$Neurona\ Oculta = \sum_{i=1}^{n} x_i \cdot w_i + b$$

Donde:
xi = Valor de la entrada.
wi= Peso de la entrada.
b= Bias

(Fórmula 10: Neurona Oculta)

Desde una perspectiva de mayor abstracción una red neuronal actuaría de forma parecida a la concatenación de varias regresiones. El resultado final de esta concatenación daría como resultado la ecuación de una única recta. Es decir, un resultado lineal (Dot, 2018b). Obtener este tipo de resultados puede ser perjudicial para el aprendizaje de nuestra red neuronal, ya que sólo estaría entrenada para resolver problemas de este formato (Dot, 2018b).

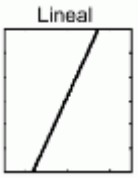

Ilustración 11: Función de activación lineal (Requena, d.f)

Para evitar este problema aparecen las funciones de activación.

La función de activación tiene como objetivo eliminar la linealidad transformando la salida de la ecuación (Dot, 2018b). Como su nombre indica esta función se encarga de indicar si una neurona se activa o no.

2.4.2.5.1. Función de activación sigmoidea

$$y = \frac{1}{1 - e^{-x}}$$

(Fórmula 11: Función de activación sigmoidea)

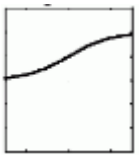

Ilustración 12: Función de activación sigmoidea (Requena, d.f)

2.4.2.5.2. Función de activación escalonada

$$y = \begin{cases} 0 \ si \ x \ < 0 \\ 1 \ si \ x \ \geq 1 \end{cases}$$

(Fórmula 12: Función de activación escalonada)

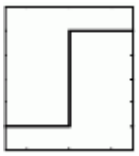

Ilustración 13: Función de activación escalonada (Requena, d.f)

2.4.1.5.3. Otras funciones de activación

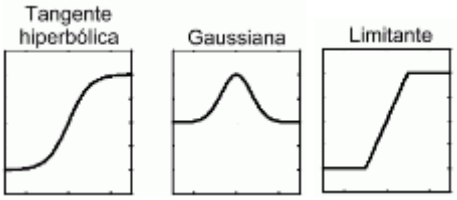

Ilustración 14. Otras funciones de activación (Requena, d.f)

2.4.2.5.4. Entrenamiento de la red neuronal

Por naturaleza, una red neuronal debe ajustarse de forma automática de cara a mejorar sus predicciones. Esto se realizará sobre los pesos de las diferentes entradas de las neuronas y otros parámetros como el bias. Iterativamente la red neuronal va calculando los valores de las neuronas capa por capa hasta obtener una salida (Dot, 2018c). De la comparación entre esa salida y la salida real obtendrá el error y será enviado hacia las capas anteriores con el objetivo de ajustar los parámetros anteriormente nombrados. Este proceso se conoce como back-propagation (Dot, 2018c).

2.4.2.5.5. Tipos de Redes Neuronales

Encontramos los siguientes tipos de redes neuronales (Pai, 2023):

- Monocapa: Sólo cuentan con una capa oculta.
- Multicapa (Multi Layer Perceptron): Cuentan con varias capas ocultas.
- Convolucionales: Utilizadas para la clasificación de imágenes, gozan de gran facilidad para simplificar o inferior las propiedades de las imágenes (Pai, 2023).
- Recurrentes: Utilizadas para la clasificación de texto, tienen en cuenta el contexto de la palabra (Pai, 2023).

Característica	Perceptrón Multicapa (MLP)	Red Neuronal Recurrente (RNN)	Red Neuronal Convolucional (CNN)
Conexiones Recurrentes	No	Si	No
Tipo de datos	Vectores bidimensionales	Texto	Imágenes
Compartición de parámetros	No	Si	Si

Tabla 10: Comparación de Redes Neuronales (Pai, 2023)

2.5. Transformers

Los transformers son un nuevo tipo de red neuronal. Surgen de la dificultad de las redes neuronales recurrentes (RNN) para tratar textos de gran longitud en términos computacionales tanto como en términos de eficiencia de las predicciones. En los textos muy largos, aparte de ser muy costosos, las palabras del inicio llegaban a perderse por ir reduciéndose su peso con el paso de los entrenamientos (Dot, 2021).

Aparecen en 2017 en la investigación publicada “Attention is All You Need” (Vaswani, 2017) El diagrama de su estructura es el siguiente:

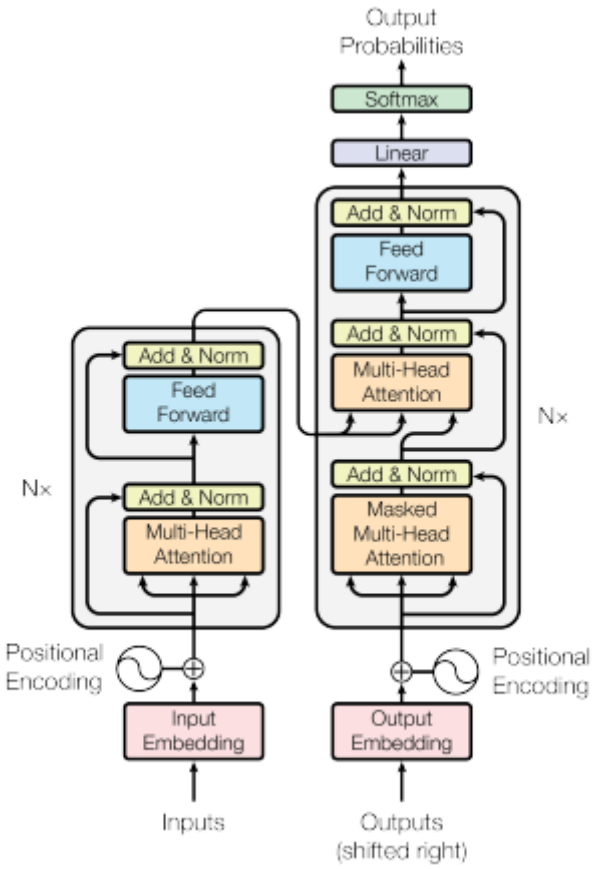

Ilustración 15: Estructura de los transformers (Vaswani 2017)

A continuación una breve descripción resumida del funcionamiento de los modelos transformers (Na, 2022).

2.5.1. Input Embedding

En primer lugar se realiza un word embedding del texto de entrada, convirtiéndo así cada palabra en un vector de números (Na, 2022).

2.5.2. Output Embedding

Es el mismo procedimiento pero con la salida del decodificador (Na, 2022).

2.5.3. Positional Encoding

Posteriormente se pasarán estos vectores por la codificación posicional. En esta fase se registra la situación de cada palabra mediante un vector en el que cada índice será calculado con una función sinusoidal (Na, 2022). La aplicación de este tipo de función nace del seguimiento de los patrones encontrados en la representación binaria numérica de cada posición. (Dot, 2021). Partiendo del siguiente ejemplo binario:

Posición n -> {p1,p2,p3,p4}

Posición 0 -> (0,0,0,0)

Posición 1 -> (0,0,0,1)

Posición 2 -> (0,0,1,0)

Posición 3 -> (0,0,1,1)

Posición 4 -> (0,1,0,0)

Podemos observar que por cada elemento de cada posición en binario se repite un cierto ciclo repetitivo. Por ejemplo:

p4: (0,1,0,1,0,1….)

p3: (0,0,1,1,0,0….)

Este tipo de fluctuación repetitiva en pulsos inspira el uso de la función sigmoidea para codificar los vectores de posición, ya que estas funciones también son cíclicas (Dot, 2021).

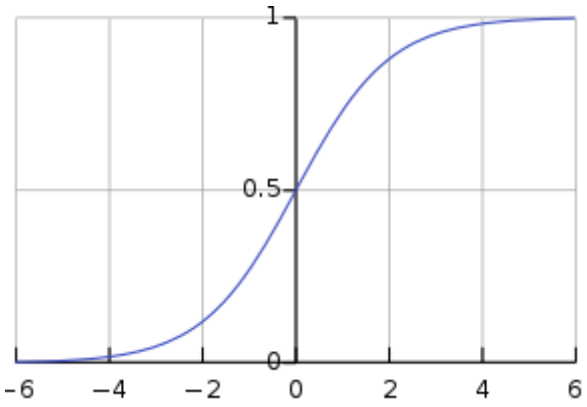

Ilustración 16: Función Sigmoidea (Wikipedia, 2023b)

Tal y como se muestra en el paper, la función es la siguiente (Vaswani, 2021). En los índices pares del vector de positional encoding se aplicará la primera función y en las impares la segunda:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Donde:
pos = Posición en el vector de input
i= Variable para iterar durante el llenado, i<=d/2, ya que por cada iteración se rellenan dos filas
D= Dimensión de la salida

(Fórmula 13: Positional Encoding (Vaswani, 2021))

2.5.4. Elementos del Decoder y Encoder

En la estructura transformers encontramos un Codificador y un Decodificador en total hay 6 de cada uno y trabajan con lotes de 512 de tamaño (Vaswani, 2017).

2.5.4.1. Feed Foward

Se trata de una red neuronal de varias capas. Se encarga de eliminar la linealidad de cara a usarlas en los siguientes procedimientos (Dot, 2021).

2.5.4.2. Add & Norm

En esta fase se suma la salida de una capa con la entrada de la capa o fase con la que está conectada y se normaliza (Na, 2022).

2.5.4.3. Máscara de Atención

Con la máscara de atención se pretende hacer que el modelo sepa qué interés tiene cada palabra, permitiendo así decidir a cuáles se le presta atención (Na, 2022). La atención es calculada con la siguiente fórmula (Vaswani, 2017):

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Donde:
Q = Matriz Query
K= Matriz Key, se traspone
V= Matriz Value
dk= Dimensión de Q y K

(Fórmula 14: Máscara de Atención (Vaswani, 2017)

La matriz Key es una matriz con los embeddings de cada palabra, es decir las características de la palabra en relación al contexto de la oración, tipología de palabra, etc… (Nabil, 2023)

La matriz Query se trata de una matriz con embeddings de características de otras palabras que podrían ser interesantes para esta (Nabil, 2023).

La matriz Value es la matriz con los embeddings de las palabras en sentido general fuera del contexto de la oración, más centrada en la información de la palabra (Nabil, 2023).

Como podemos observar la piedra angular de este cálculo de la atención se obtiene del producto vectorial de las matrices Query y Key. Por lo que cuanto más cercanos sean los vectores de embeddings mayor será la atención de la palabra. Se disponen de tres variantes de este cálculo de la atención.

Más en concreto la filosofía transformers utiliza este mecanismo de una forma iterativa dividiendo el conjunto, conocida como la atención multicapa. En este crea una serie de subgrupos y finalmente se calcula la media para cada palabra (Na, 2022).

2.5.5.Encoder

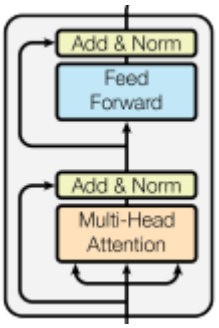

Ilustración 17: Encoder de un Transformers (Vaswani 2017)

En el encoder encontramos una capa de asignación de atención, seguida de una de normalización posteriormente comunicada con la red neuronal feed foward y su respectiva capa de normalización.

2.5.5. Decoder

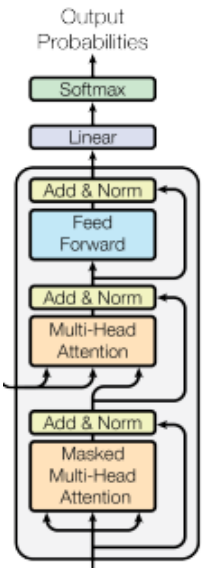

Ilustración 18: Decoder de un Transformers (Vaswani 2017)

En el decoder encontramos casi los mismos elementos que en el Encoder. Sin embargo, la primera capa multi head de atención es enmascarada. Es el mismo proceder que en el mecanismo de atención explicado anteriormente, pero con una sutil diferencia. Al procesar se establecen algunas posiciones de la matriz como desconocidas o enmascaradas para evitar usar valores de atención de otras posiciones de otras iteraciones diferentes a la actual que podrían influir. Este enmascaramiento oculta una mitad triangular de la matriz (Na, 2022).

Esto es debido a que al tener acceso a la atención entre cada palabra en una matriz. Si estamos accediendo a la atención de la *palabra1* con la *palabra2* podremos acceder a la información de la atención de la palabra3 con la palabra1. Se estaría adelantando información que debería ser desconocida en la actual iteración y que podría influir en el resultado (Na, 2022). Posteriormente encontramos las mismas capas que en el encoder.

Una vez realizada la decodificación pasamos a una capa en la que se transforma la salida con una función softmax, es decir, se aplica un producto vectorial para obtener probabilidades entre 0 y 1 (Na, 2022). Por ejemplo, para una clasificación se obtendrá la probabilidad de que pertenezca a cada etiqueta.

2.6. Transformers BERT

Los modelos BERT son los Bidirectional Encoders from Transformers. Son un modelo del lenguaje basado en la filosofía transformers. La principal característica es que utilizan el contexto de la palabra tanto a izquierda como a derecha. Siguen una estructura parecida a los transformers, sin embargo estos no usan decoders y tienen algunas modificaciones en los parámetros (Nbro, 2020)

Se distinguen dos tipologías de entrenamiento en los modelos BERT. Encontramos el entrenamiento y el fine tuning o ajuste fino.

En el entrenamiento se educa el modelo desde 0 para un determinado idioma con una serie de recursos lingüísticos como pueden ser ontologías, diccionarios, etc…

Un ajuste fino consiste en el entrenamiento de un modelo pre entrenado con un corpus específico para una temática en especial.

Algunos de los principales modelos BERT estudiados son:

2.6.1. BERT-Base

Es el modelo original sobre el que beben todos los demás. Podemos encontrar las siguientes variantes:

2.6.1.1. Uncased

Este modelo BERT no diferencia entre las palabras mayúsculas y minúsculas en el entrenamiento (Huggin Face, s.fa.).

2.6.1.2. Cased

Este modelo BERT sí diferencia entre las palabras mayúsculas y minúsculas en el entrenamiento (Huggin Face, s.fb.).

2.6.2. Distil-BERT-Base

Es una variación de BERT original entrenado con casi la mitad del corpus y más rápido. Su rendimiento sólo baja un 5 % en comparación con el anterior BERT (Huggin Face, s.fc.). También cuenta con versiones Cased y Uncased.

2.6.3. Beto

Es un modelo bert con tamaño de gran parecido a bert básico entrenado con gran cantidad de líneas de texto en español de numerosas fuentes y recursos lingüísticos (GitHub, s.fa.).

2.6.4. Maria

Este modelo nace del proyecto de investigación homónimo para la generación de resúmenes, compresión y escritura de texto en español. Fue entrenado con más de 200 millones de documentos (GitHub, s.fb.).

2.6.5. Bertin

Es un modelo basado en el modelo BERT RoBERTa realizado durante el evento Flax/Jax community en el que Google cedió unas unidades de procesamiento (Huggin Face, s.fd.).

2.6.6. Multilingual-BERT-base

Este modelo fue pre entrenado con una cantidad de más de cien idiomas entre los que se incluye el español (Huggin Face, s.fe.). Este BERT cuenta con su versión distil o reducida así como con su versión uncased y cased.

2.7. Métricas

Es necesario establecer una serie de métricas para la validación del rendimiento de los modelos entrenados con la finalidad de elegir a los mejores y valorar el funcionamiento de estos (Heras, 2020b).

Mediante la siguiente matriz de cara al desarrollo de las métricas, donde por cada etiqueta a evaluar se irá midiendo la eficacia del modelo. Si estamos evaluando la clase 1, esta será definida como clase A mientras que las n restantes serán denominadas como pertenecientes a la clase B (Heras, 2020b).

	Etiqueta	**Predicha**
Etiqueta Real	**A**	**B**
A	VP	FN
B	FP	VN

Tabla 11: Matriz de confusión para la clase a evaluar (Heras, 2020b)

Donde:

- **A** es la etiqueta que estamos sometiendo a evaluación.
- **B** es la etiqueta contraria a la que estamos sometiendo a evaluación.
- **VN:** Verdaderos Negativos. Una predicción es un verdadero negativo cuando la etiqueta real es de tipo B y la etiqueta predicha es de tipo B.
- **FP:** Falsos Positivos. Una predicción es un falso positivo cuando la etiqueta real es de tipo B y la etiqueta predicha es de tipo A.
- **FN:** Falsos Negativos. Una predicción es un falso negativo cuando la etiqueta real es de tipo A y la etiqueta predicha es de tipo B.
- **FN:** Verdaderos Positivos. Una predicción es un verdadero positivo cuando la etiqueta real es de tipo A y la etiqueta predicha es de tipo A.

Por cada etiqueta que nuestro modelo es capaz de predecir y en base a la tabla anterior el cálculo de las métricas se realiza de la siguiente forma:

2.7.1. Precision

La precisión mide la capacidad del modelo para detectar positivos correctamente. Se calcula mediante el cociente de los verdaderos positivos con la suma de los verdaderos positivos y los falsos positivos (Heras, 2020b):

$$Precisión = \frac{VerdaderosPositivos}{VerdaderosPositivos \ + \ FalsosPositivos}$$

(Fórmula 15: Precisión)

2.7.2. Recall

El recall mide la cantidad de etiquetas de la clase que estamos evaluando es capaz de predecir nuestro modelo. Se calcula mediante el cociente de los verdaderos positivos con la suma de los verdaderos positivos y los falsos negativos (Heras, 2020b):

$$Recall = \frac{VerdaderosPositivos}{VerdaderosPositivos \ + \ FalsosNegativos}$$

(Fórmula 16: Recall)

2.7.3. F1-score

El f1-score se corresponde con la media entre precisión y el recall pero de forma armónica. En primer lugar, se realiza el cociente entre la multiplicación de la precisión por el recall y la suma entre la precisión y el recall. Posteriormente, el resultado obtenido se multiplica por dos (Heras, 2020b). Esta métrica es de destacada utilidad, ya que solo obtendrá valores altos si tanto la precisión como el recall lo son. Por otro lado, si cualquiera de los dos es bajo el f1_score también lo será (Heras, 2020b).

$$F1\_Score = 2 \cdot \frac{Precisión \cdot Recall}{Precisión + Recall}$$

(Fórmula 17: F1-Score)

2.7.3. Accuracy

Esta métrica calcula cuántas etiquetas predice correctamente el modelo. Da lugar a sobreestimaciones en los resultados si no se comparada con otras métricas. Se obtiene con la siguiente fórmula (Heras, 2020b):

$$Acurracy = \frac{VerdaderoPositivos + VerdaderosoNegativos}{VerdaderoPositivos + VerdaderosoNegativos + FalsosPositivos + FalsosNegativos}$$

(Fórmula 18: Accuracy)

2.7.4. Support

El support se corresponde con él número de elementos de cada etiqueta que son verdaderas (Sk Learn, s.f.). Por ejemplo, si de 100 elementos que vamos a utilizar en la validación 55 son de la etiqueta 1 y 45 de la etiqueta 2 obtendremos el siguiente support:

$$Peso\ etiqueta\ = \frac{Support\ Etiqueta}{Support\ Total}$$

(Fórmula 20: Weighted Average)

Etiquetas	Support
Etiqueta 1	55
Etiqueta 2	45

Tabla 12: Ejemplo de Support

2.7.5. Macro Average

El macro average consiste en realizar una media sin tener en cuenta el peso de cada una (Sefidian, 2023). Encontramos tanto el macro average precision, macro average recall como el macro average recall. Cabe destacar que para el marco support, simplemente se realizará una suma del support de las dos etiquetas.

$$Macro\ avg\ Métrica = \frac{Métrica\ etiqueta\ 1 + \ldots\ldots + Métrica\ etiqueta\ n}{n}$$

(Fórmula 19: Macro average)

Etiquetas	precision	recall	f1-score	Support
Etiqueta 1	0.89	1.0	0.943333	36
Etiqueta 2	1.0	0.823333	0.903333	26
Macro avg	**0.946667**	**0.913333**	**0.920000**	**62**

Tabla 13: Ejemplo de Macro Average

2.7.6. Weighted Average

En el weighted average si se realiza una media teniendo en cuenta el peso. Este peso será asignado según el número de instancias de cada etiqueta sobre todas las métricas de cada clase (Sefidian, 2023). El cálculo quedaría de la siguiente forma:

$$Weighted\ avg\ Métrica = \frac{Métrica\ etiqueta\ 1 \cdot Peso\ etiqueta\ 1 + \ldots\ldots\ldots + \ Métrica\ \ Etiqueta\ \ n \cdot Peso\ etiqueta\ \ n}{n}$$

(Fórmula 20: Weighted average)

Un ejemplo de resultados obtenidos sería:

Etiquetas	precision	recall	f1-score	Support
Etiqueta 1	0.89	1.0	0.943333	36
Etiqueta 2	1.0	0.823333	0.903333	26
Weigthted avg	**0.936667**	**0.926667**	**0.923333**	**62**

Tabla 14: Ejemplo de Weighted Average

2.8. Validación Cruzada

La validación cruzada es un método para evaluar el rendimiento del modelo. Se realizan una serie de iteraciones en las que van cambiando el conjunto con el que el modelo es entrenado y el conjunto con el que el modelo es validado. Estos dos conjuntos son excluyentes, es decir, un elemento del conjunto de entrenamiento no puede pertenecer al conjunto de elementos de validación. Se han estudiado los siguientes métodos para la validación cruzada:

2.8.1. Validación K-Fold

El Corpus se divide en *k* subconjuntos. Por lo que obtendremos un número *k* de iteraciones. En cada una de ellas se entrenará el modelo con k-1 conjuntos y se validará con el conjunto restante (Schmid, 2020). Por ejemplo, si realizamos una validación cruzada k-fold donde la *k* es 3 encontraremos la siguiente división:

- Sea un conjunto de archivos: [1,2,3,4,5,6,7,8,9] donde k=3:
- Subconjuntos: {1,3,5} {7,8,9} {2,4,6}
- Iteraciones:
    - Iteración 1: Entrenamiento: {1,3,5} {7,8,9} Validación: {2,4,6}
    - Iteración 2: Entrenamiento: {1,3,5} {2,4,6} Validación: {7,8,9}
    - Iteración 3: Entrenamiento: {2,4,6} {7,8,9} Validación: {1,3,5}

2.8.2. Validación Leave-one

En este tipo de validación cruzada el se utiliza como conjunto de entrenamiento todo el corpus menos una unidad, en este caso un documento de texto. Este archivo restante será utilizado para la validación del modelo (Wikipedia, 2023c). Ejemplo demostrativo:

- Sea un conjunto de archivos: [1,2,3,4]
- Iteraciones:
  - Iteración 1: Entrenamiento: {1,2,3} Validación: {4}
  - Iteración 2: Entrenamiento: {1,3,4} Validación: {2}
  - Iteración 3: Entrenamiento: {2,3,4} Validación: {1}
  - Iteración 4: Entrenamiento: {1,2,4} Validación: {3}

Para este proyecto nos quedaremos con una **validación cruzada K-Fold con k=3**. Esta decisión de diseño está fundada por el tamaño del corpus.

3. ESTUDIO DE LAS TECNOLOGÍAS EXISTENTES Y NECESARIAS: DESARROLLO DEL PROYECTO Y MEMORIA

Una vez realizada una investigación del estado de la cuestión de los sistemas clasificadores y diferentes modelos, se procedió a investigar las tecnologías con las que implementar la aplicación.

3.1. Lenguaje de programación: Python

Ilustración 19: Logo de Python

La elección del lenguaje de programación con el que trabajar para la resolución de este problema fue relativamente rápida. Python se trata de un lenguaje de programación muy utilizado en los campos de Procesamiento del Lenguaje Natural, Big Data y Ciencia de Datos, entre muchos otros.

Es muy intuitivo, fácil de usar y fácil de entender. Además, se presenta muy legible, con gran compatibilidad y facilidad para la integración de bibliotecas o librerías externas con un simple comando de código. Estas librerías presentan una documentación muy completa totalmente interpretable a diferencia de otros lenguajes de programación.

Cabe destacar que la gran mayoría de los elementos estudiados en el punto 2 como todas las tecnologías conocidas están implementados en Python y son accesibles mediante sus librerías.

3.2. Entorno de programación: PyCharm

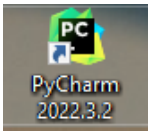

Ilustración 20: Logo de PyCharm

PyCharm es un IDE de programación para Python de Jetbrains. Esta empresa presenta una gran colección de entornos de programación accesibles. En este caso se accedió a la versión completa con la licencia educativa de la Universidad de Jaén.

El hecho de haber probado y tener experiencia en otros IDE de la misma familia en otras asignaturas fue un factor decisivo para elegir este entorno de trabajo. Se trata de un entorno bastante intuitivo y que beneficia en gran medida la productividad del desarrollador mediante sus asistentes y plugins.

Sin bien es cierto que podemos encontrar otras alternativas a PyCharm como Visual Studio o Google Colab, finalmente fueron descartados.

Visual Studio es bastante parecido, pero como desarrollador estaba más familiarizado con PyCharm. Por otro lado, se encontraba Google Colab. Aunque fue usado en la asignatura Procesamiento del Lenguaje Natural, cuenta con ciertas limitaciones y está más pensado para un entorno colaborativo. El acceso a los archivos del escritorio es algo tedioso, los procesamientos en la nube en ocasiones dan problemas en términos de cómputo y es poco intuitivo para la creación de proyectos de gran tamaño con multitud de scripts.

En base a todos estas circunstancias fue elegido finalmente PyCharm.

3.3. Framework web: Flask

Ilustración 21: Logo de Flask

Flask se trata de un framework para el desarrollo de aplicaciones web mediante Python. Este marco de trabajo es bastante sencillo de utilizar y está orientado al desarrollo de pequeñas aplicaciones web de poca complejidad.

Podemos encontrar otros competidores como Django. A continuación, un pequeño estudio de las diferencias mediante una comparación entre Flask y Django:

Modelo	Flask	Django
Curva de Aprendizaje	Baja	Alta
Flexibilidad	Alta	Baja
Compatibilidad con otros frameworks	Baja	Alta
Integración con bases de datos	Media	Alta
Mapeo Objeto Relacional	Externo	Propio
Documentación precisa	Alta	Medio
Servidor Incluido	Si	No

Tabla 15: Comparativa entre Flask y Django (IONOS, 2022)

Tras analizar las principales virtudes y desventajas de cada framework se acabó eligiendo Flask por los siguientes motivos:

- Se trata de una página web sencilla.
- Flask provee de un servidor para lanzar la ejecución incluido.
- Es un framework con una curva de aprendizaje muy baja y es muy flexible.
- La documentación de Flask es más accesible, sencilla e intuitiva

3.3. Framework de plantillas: Bootstrap

Ilustración 22: Logo de Bootstrap

Para el diseño Front-end de la aplicación web se ha optado por este framework. Es de gran utilidad, ya que ofrece una serie de funciones creadas que ahorran mucho tiempo en el desarrollo CSS y demás elementos de la capa de visualización.

Es de gran utilidad para desarrollar vistas de aplicaciones que se integren perfectamente con los diferentes navegadores y dispositivos. Por otro lado, dispone de una documentación bastante extensa y precisa. Además, hay una gran comunidad de desarrolladores que lo utilizan.

3.4. Plataforma de creación de diagramas: Diagrams.net

Ilustración 23: Logo de Diagrams.net

Se utilizó esta herramienta para la realización de esos estudios previos de las tecnologías, su posterior reflejo en la memoria del proyecto, así como la generación de los diagramas para la implementación del proyecto era necesaria una aplicación para la creación de diagramas.

Diagrams.net, también conocido como Draw.io, cuenta con gran variedad de diagramas y usa almacenamiento en la nube integrado con Google y cuenta con gran flexibilidad para la descarga en diferentes formatos.

Fue elegido al contar con bastante experiencia en el uso de este para otras asignaturas.

3.5. Plataforma de creación de diagramas de Gantt: Canva

Ilustración 24: Logo de Canva

Esta página cuenta con varias plantillas para realizar diagramas de Gantt así como otro muchos tipos de representaciones visuales.

La edición del propio diagrama de Gantt es bastante sencilla e intuitiva a diferencia de otros portales de creación de herramientas del mismo tipo en los que la generación de estos esquemas era un proceso bastante tardío y engorroso.

Es de acceso gratuito y cuenta con almacenamiento en la nube.

3.6. Generador de fórmulas matemáticas: iMathEQ

Ilustración 25: Logo de iMathEQ

La gran mayoría de las ecuaciones de los puntos anteriores han sido generadas con este sitio web. Se trata de una página bastante sencilla y simple de usar.

Cuenta con una amplia variedad de conceptos matemáticos y es accesible online, sin necesidad de instalación en la computadora a diferencia de otros generadores de fórmulas.

3.7. Generador de diagramas de redes neuronales: http://alexlenail.me/NN-SVG/

Ilustración 26: Logo de NN-SVG

La red neuronal del punto 2.4.2.5. fue realizada en este espacio web. Al no disponer de ejemplos de redes neuronales del todo orientativas se optó por la generación propia de un ejemplo.

3.8. Repositorio utilizado: Github

Ilustración 27: Logo de Github

Se trata de un repositorio y gestor de versiones que hemos ido utilizando y aprendiendo durante la carrera, por lo que no iba a quedar fuera para la realización del Trabajo de Fin de Grado.

Es bastante simple de usar en su versión de escritorio, nos ahorra utilizar toda la familia de comandos propios de Git. Permite la creación de ramas, rápida edición de archivos y muchas otras facetas propias de los gestores de versiones.

3.9. Editor de texto para la memoria: Google Docs

Ilustración 28: Logo de Google Docs

Este editor de texto es claro ganador en la comparativa contra su competencia. Es gratuito y tiene soporte en la nube mediante Google Drive. La edición en el archivo se guarda en tiempo real y tiene una integración completa con todos los programas de la familia Google.

Además, su acceso es extremadamente intuitivo y sin necesidad de instalar nada, en comparación con otras aplicaciones como la versión gratuita de Microsoft Word.

4. DISEÑO DE LA APLICACIÓN

El diseño de este proyecto se divide en dos partes diferenciadas, las cuales se detallarán a continuación.

En la primera parte, una vez completado el estudio previo de las diferentes tecnologías, encontramos el diseño de un modelo clasificador que hace uso de modelos BERT y de modelos que no son BERT.

Se hará uso de estos dos tipos de modelos de cara a la etapa de evaluación y comparación de resultados. Se someterá a todos los modelos a una validación cruzada k-fold para comprobar el rendimiento mediante las métricas estudiadas y el tiempo de entrenamiento.

En la segunda parte del desarrollo, se trabajará en la creación de la página web sencilla para la visualización de resultados. En esta página el usuario podrá escribir un texto o subir un documento *.docx*. Posteriormente, mediante un botón, el documento será clasificado como delito o como delito de no odio. Seguidamente, la vista web será actualidad con el resultado. Ha de ser una página web intuitiva y visualmente vistosa. Esta página web será implementada con una filosofía Modelo-Vista-Controlador logrando así el desacoplamiento de los diferentes componentes.

Una vez implementadas ambas partes, se procederá a un análisis de los resultados obtenidos por los diferentes modelos de cara a seleccionar los mejores para ser accesibles desde la página web.

4.1. Requisitos

En esta sección se procederá a detallar todos los requisitos funcionales y no funcionales. Es decir, el conjunto de requerimientos que debe cumplir el sistema a desarrollar. Los requisitos funcionales indican qué debe hacer el sistema, mientras que los no funcionales nos ilustran el cumplimiento de unos parámetros mínimos o máximos como el rendimiento, usabilidad, velocidad de ejecución, tiempo de espera, etc…

4.1.1. Requisitos Funcionales

Requisitos Funcionales
RF1: El sistema debe clasificar una denuncia policial.
RF2: El sistema debe estar entrenado con el corpus de atestados policiales.
RF3: El sistema debe disponer de la capacidad de leer y procesar las denuncias policiales del corpus.
RF4: El sistema debe contar con la capacidad de entrenar diferentes modelos de clasificación.
RF5: El sistema debe tener un validador que genere unas métricas para medir el rendimiento de los diferentes modelos.
RF6: El sistema debe contar con una forma de selección de entre todos los modelos entrenados disponibles.
RF7: El sistema debe contar con una página web para la visualización de los resultados.

Tabla 16: Requisitos Funcionales

4.1.2. Requisitos No Funcionales

Requisitos No Funcionales
RNF1: La interfaz del sistema debe cumplir los principios de usabilidad.
RNF2: El tiempo de respuesta para la clasificación de una denuncia debe ser inferior a 4 segundos.
RNF3: El tiempo de respuesta para mostrar un mensaje de error en la interfaz web debe ser instantáneo.
RNF4: El espacio ocupado por parte de los modelos entrenados debe ser inferior a 10 GB
RNF5: El sistema debe contar con una interfaz amigable e intuitiva.
RNF6: El tiempo de entrenamiento de un modelo debe ser inferior a 5 horas.
RNF7: El tiempo de respuesta para cambiar de una vista de la web a otra debe ser menor a 1 segundo.

RNF8: La vista web debe hacer uso de metáforas como botones, ventanas de escritura, etc…
RNF9: La página web no tendrá tolerancia a fallo durante la subida o escritura de un archivo en formato incorrecto, notificándose así al usuario.
RNF10: El sistema clasificador no tendrá tolerancia al fallo durante un entrenamiento o una validación, notificandolo así por pantalla y deteniendo la ejecución
RNF11: Las vistas web han de ser web responsive para facilitar su uso y accesibilidad.
RNF12: El tiempo de lectura y procesado de denuncias debe ser menor que el de entrenamiento.

Tabla 17: Requisitos No Funcionales

4.2. Historias de Usuario

A continuación, las historias de usuario. En este apartado se visualizará en una serie de mini tarjetas en formato tabla todos los roles del software desarrollado para cada usuario junto con las respectivas funciones y cometidos finales de estas. Principalmente, distinguimos dos tipos de usuarios: El usuario normal que hará uso de la web y el analista programador que hará uso del propio sistema a niveles internos de entrenamiento y medición de rendimiento.

Como	**Usuario**
Quiero	Poder acceder a una página web
Para	Visualizar los resultados de la clasificación

Tabla 18: Historia de Usuario 1

Como	**Usuario**
Quiero	Poder escribir un texto en la página web
Para	Clasificar una denuncia policial en formato escrito

Tabla 19: Historia de Usuario 2

Como	**Usuario**
Quiero	Poder subir un texto a la página web
Para	Clasificar una denuncia en formato docx

Tabla 20: Historia de Usuario 3

Como	**Usuario**
Quiero	Poder seleccionar el tipo de modelo a usar en la clasificación.
Para	Para elegir el modelo que crea más conveniente

Tabla 21: Historia de Usuario 4

Como	**Usuario**
Quiero	Poder cambiar entre ambos formatos de clasificación (archivo o texto)
Para	Tener libertad de elección

Tabla 22: Historia de Usuario 5

Como	**Analista Programador**
Quiero	Hacer el fine-tuning de un modelo BERT
Para	Poder clasificar denuncias policiales de delitos de odio

Tabla 23: Historia de Usuario 6

Como	**Analista Programador**
Quiero	Realizar una validación cruzada de varios modelos
Para	Comprobar cuáles funcionan mejor

Tabla 24: Historia de Usuario 7

Como	**Analista Programador**
Quiero	Entrenar modelos distintos a BERT
Para	Realizar comparaciones

Tabla 25: Historia de Usuario 8

Como el proyecto está divido en la implementación de una web y de un sistema clasificador dividiremos el diseño en dos principales apartados: El diseño del sistema clasificador y el diseño de la página web.

4.3. Diseño del sistema clasificador

Se trata de la aplicación que nos servirá para entrenar los modelos y para realizar validaciones cruzadas de estos. No tendrá interfaz propia más allá de la propia del entorno de desarrollo.

4.3.1. Diagram de clases

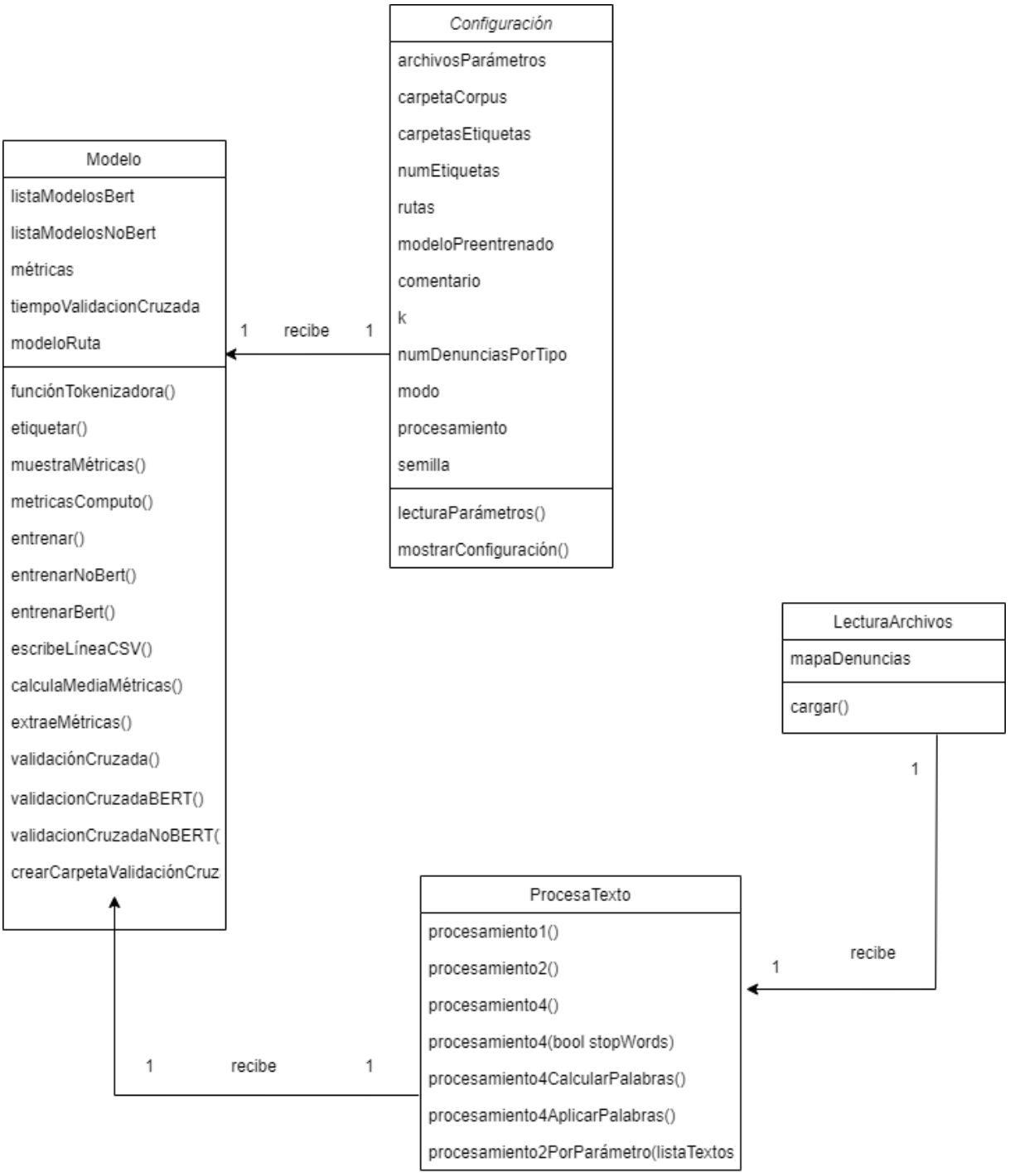

Ilustración 29: Diagrama de Clases del Sistema Clasificador

En la estructura general del sistema encontramos la clase modelo, la cual será el epicentro del clasificador. Esta clase recibirá los textos procesados y la información del fichero de configuración.

La clase *Configuración* se encargará de la lectura del fichero de configuración y la clase *Procesatexto* se encargará de procesar los archivos leídos por la clase *LecturaArchivos*.

4.3.1.1 Clase Modelo

Se trata de la clase del modelo de clasificación.

Atributos:

- **ListaModelosBert:** Lista con todos los modelos BERT que son elegibles desde el fichero de configuración.
- **ListaModelosNoBert:** Lista con los modelos que no son BERT elegibles en el fichero de configuración.
- **Métricas:** Lista para guardar el resultado de todas las métricas de una validación cruzada.
- **TiempoValidaciónCruzada:** Variable para guardar el tiempo total de la validación cruzada.
- **ModeloRuta:** Ruta en la que guardar el modelo.

Funciones:

- **FunciónTokenizadora():** Función para tokenizar los textos.
- **Etiquetar():** Función para crear un diccionario que contiene el texto del documento y su categoría.
- **MuestraMétricas():** Función para mostrar las métricas.
- **MétricasConputo():** Función para calcular métricas en mitad de una ejecución de entrenamiento.
- **Entrenar():** Función que llamará a *entrenarBERT* o a *entrenarNoBERT* según el modelo introducido en el archivo de configuración.
- **EntrenarBERT():** Función que contará con las pautas de entrenamiento de los modelos que son BERT.
- **EntrenarNoBERT():** Función que contará con las pautas de entrenamiento de los modelos Sk-Learn
- **EscribeLíneaCSV():** Función para escribir una línea en un archivo *.csv*.
- **CalculaMediaMétricas():** Función para calcular la media de las diferentes métricas de una validación cruzada.
- **ExtraeMétricas():** Función para convertir una métrica en formato vector.
- **ValidaciónCruzada():** Función para elegir que tipo de validación cruzada efectuamos según el modelo del fichero de parámetros.
- **CrossValidationBERT():** Función para realizar una validación cruzada de un modelo BERT.
- **CrossValidationNoBERT():** Función para realizar una validación cruzada de un otro modelo BERT.
- **CrearCarpetaValidacionCruzada:** Función para crear una carpeta con el corpus y los archivos de validación utilizados en la última división de una validación k-fold.

4.3.1.2. Clase Configuración

En esta clase se almacenarán todos los valores introducidos en el fichero de configuración *configuración.txt.*

Atributos:

- **NumEtiquetas:** Variable para almacenar el número de etiquetas.
- **CarpetasEtiquetas:** Variable para almacenar las carpetas de las etiquetas.
- **CarpetaCorpus:** Variable para almacenar la carpeta en la que se encuentra el corpus de textos.
- **ModeloPreentrenado:** Variable para almacenar el modelo elegido.
- **ArchivosParámetros:** Variable para almacenar el nombre del archivo de parámetros.
- **Rutas:** Variable para almacenar las rutas.
- **Nombre:** Variable para almacenar un nombre identificativo de la iteración.
- **K:** Variable para guardar la *k* de la validación cruzada k-fold.
- **DenunciasPorTipo:** Variable para almacenar el número de archivos de cada etiqueta con el que se entrena el modelo.
- **Modo:** Variable para almacenar el modo de ejecución. Entrenamiento o validación.
- **Procesamiento:** Entero para indicar el número de procesamiento a usar.
- **Semilla:** Entero con la semilla para la ejecución.

Funciones:

- **LecturaParametros:** Función para extraer todos los valores de configuración del fichero *configuración.txt*.
- **MostrarConfiguración:** Función para mostrar la configuración obtenida.

4.3.1.3. Clase ProcesaTexto

Se trata de una clase usada para la validación cruzada de cara a reducir los tiempos de ejecución.

Funciones:

- **Procesamiento1():** Esta función se encarga de eliminar los espacios entre cada párrafo de los textos. Simplemente, se usa para una mejor visualización de los textos en la comparación de la estructura de las diferentes denuncias.
- **Procesamiento2():** Esta función sirve para eliminar de los textos aquellas palabras que se encuentran en mayúscula completamente, no son alfabéticas, numéricas y los signos de puntuación. Es usada para dotar de más imprecisión a los clasificadores de cara a evaluarlo con más profundidad. También realiza las funciones de *procesamiento1()*
- **Procesamiento3():** Esta función se encarga de realizar lo mismo con las anteriores, sólo que además, quita algunas frases comunes repetitivas de cada tipo de denuncia. De las denuncias de delito de odio elimina las palabras

entre “Instructor y “ocurridos”, mientras que de las que no son delitos de odio elimina las palabras entre “Atestado” y “Dependencia”. El objetivo de esta función es el mismo. Dotar de aún más imprecisión a los sistemas para poder descubrir cuáles trabajan mejor.

- **Procesamiento4():** Esta función se trata de un *procesamiento2()* que utiliza una bolsa de palabras para eliminarlas de las denuncias. En primer lugar llama al método de cálculo de palabras a eliminar y posteriormente al método de aplicación de este borrado. Estas dos funciones son explicadas a continuación.
- **Procesamiento4CalcularPalabras():** Esta función almacena las siguientes palabras en una bolsa:
    - Palabras que aparecen en todos las denuncias de Odio.
    - Palabras que aparecen en todas las denuncias de No Odio.
    - Palabras exclusivas de las denuncias de Odio.
    - Palabras exclusivas de las denuncias de No Odio.
    - Palabras que más se repiten en las denuncias de Odio.
    - Palabras que más se repiten en las denuncias de No Odio.
- **Procesamiento4AplicarPalabras():** Esta función realiza un *procesamiento2()* y seguidamente elimina todas las palabras calculadas en la función anterior.
- **Procesamiento2PorParámetro():** Se trata de un procesamiento 2 que recibe un parámetro listo para poder recibir textos desde el controlador. Sólo el procesamiento 2 cuenta con esta función porque será elegido para el entrenamiento y prueba de los modelos en base al análisis del punto 7.

4.3.1.4. LecturaArchivos

Clase para leer los archivos en formato *.docx* del corpus y almacenarlos.

Atributos:

- **MapaDenuncias:** Mapa de listas con las listas de textos de cada etiqueta.

Funciones:

- **Cargar():** Función para leer y guardar los archivos del corpus.

4.2. Diseño de la página web

4.2.1. Diagrama de la Arquitectura MVC

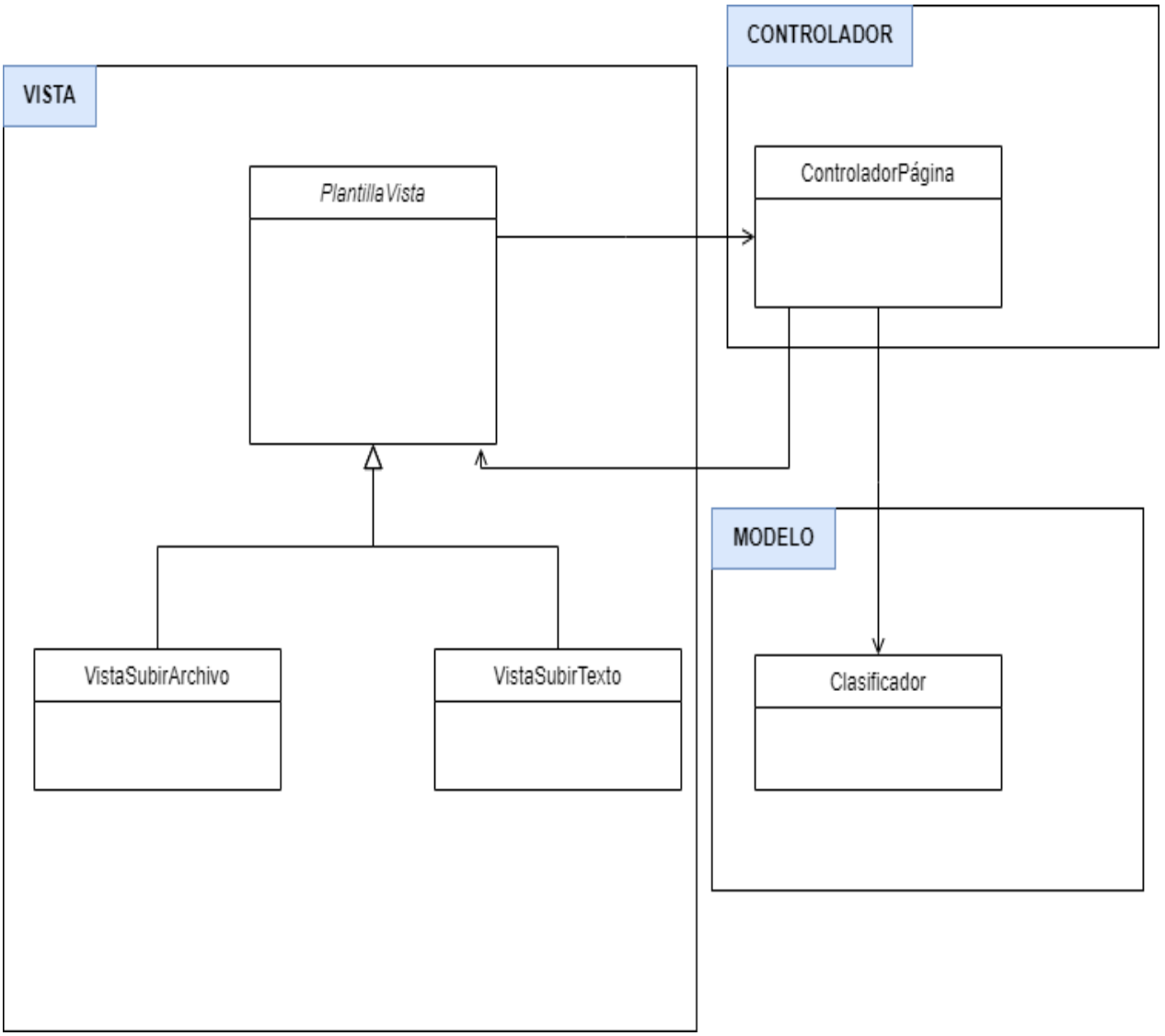

Ilustración 30: Diagrama MVC de la web

La arquitectura establecida para la página web de visualización de resultados se trata de una estructura MVC. Una estructura Modelo-Vista-Controlador en la que la vista se comunica con el controlador mediante el enrutado y el controlador realiza cambios en la vista a través del parámetro data de Flask.

El controlador de la página se encarga de enviar el texto recibido al clasificador y actualizar la vista con los datos recibidos de la clasificación en la vista.

4.2.2. Diagrama de Clases de la Web

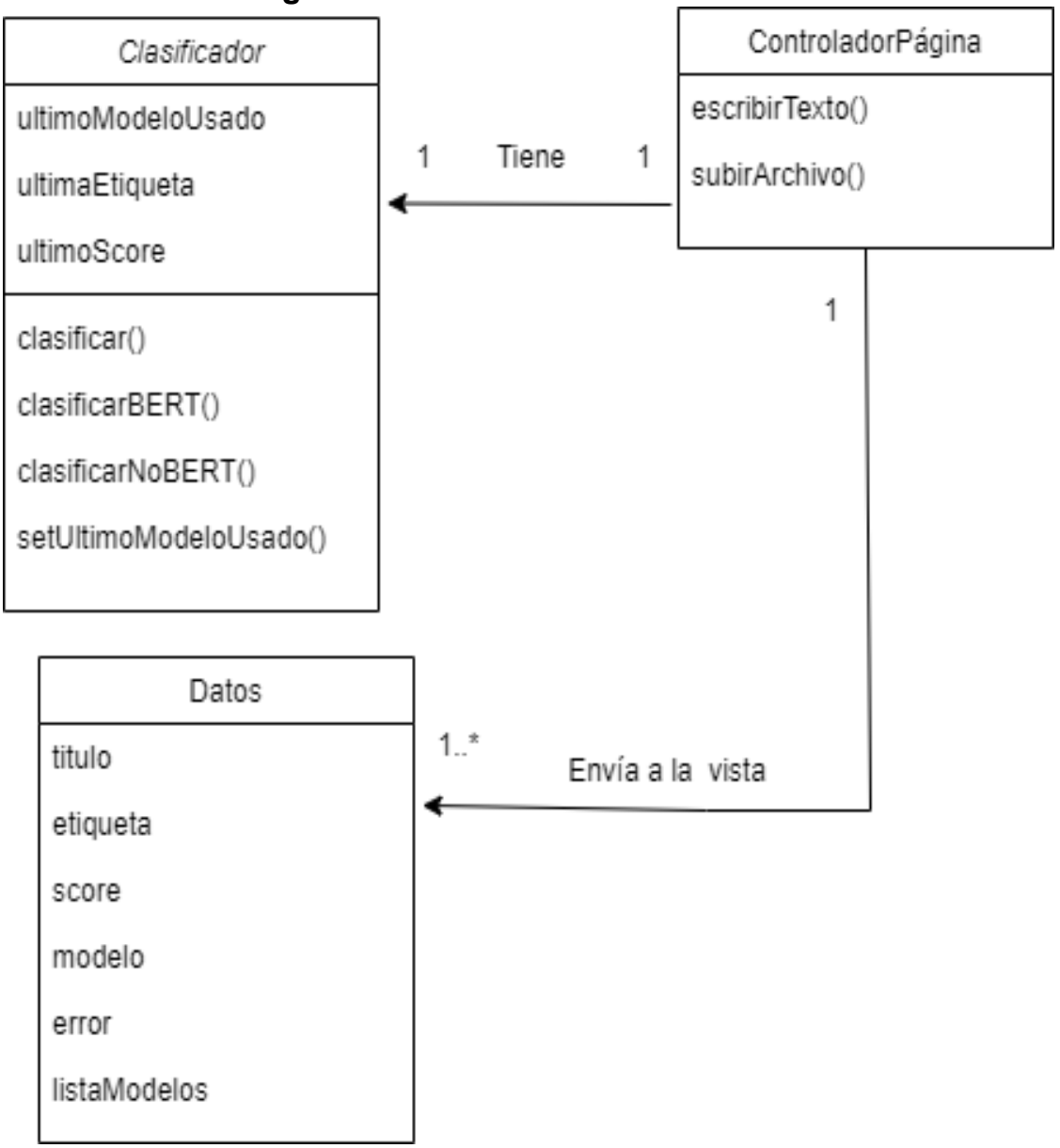

Ilustración 31: Diagrama de Clases de la Web

4.2.2.1. Clase Clasificador

Se trata de una clase para crear un clasificador accediendo al modelo BERT pre entrenado.

Atributos:

- **UltimoModeloUsado:** Variable para almacenar el último modelo que se usó en una clasificación. Es de especial utilidad, ya que en función del último modelo usado se colocará éste en la primera posición de la lista de modelos disponibles.
- **UltimaEtiqueta:** Variable para almacenar la última etiqueta asignada en una clasificación de un texto.
- **UltimoScore:** Variable para almacenar el último score de esta clasificación

Funciones:

- **clasificar():** Función para elegir el tipo de clasificación en función del modelo elegido. Según se trate de un modelo BERT o no.
- **clasificarBERT():** Función para clasificar un texto con un modelo BERT.
- **clasificarNoBERT():** Función para clasificar un texto con un modelo no BERT.
- **setÚltimoModeloUsado():** Función para establecer en primera posición de la lista de modelos el último modelo usado.

4.2.2.2. Clase ControladorPagina

Es el controlador de la página web. Se encarga de procesar todas las peticiones enviadas al enrutador establecido.

Funciones:

- **EscribirTexto():** Función que es llamada cuando el usuario rellena el campo de texto de la vista *EscribirTexto*.
- **SubirArchivo():** Función que es llamada cuando el usuario completa la subida de un archivo docx en la vista *SubirArchivo*.

4.2.2.3. Clase Datos

Esta clase es usada para enviar y recibir información de la vista

Atributos:

- **Titulo:** Variable con el título de la página.
- **Etiqueta:** Variable con la etiqueta a mostrar en el mensaje con la clasificación obtenida.
- **Score:** Variable con el score obtenido en esa clasificación.
- **Modelo:** Variable que almacena el modelo a usar en la clasificación.
- **Error:** Variable para guardar un mensaje de error. Este tiene prioridad para ser mostrado en la misma zona que el de score y modelo.
- **ListaModelos:** Lista con los diferentes modelos disponibles para clasificar.

4.2.3 Diseño del Wireframe

En primer lugar, se realizó un Wireframe de las diferentes vistas. Un wireframe es una representación esquemática visual donde prima la organización y jerarquía del contenido por encima de detalles de diseño gráfico.

Al tratarse de una página web simple, en su totalidad cuenta con dos vistas: Una vista para la clasificación de archivos y otra para la clasificación de textos.

La página se dispone con una estructura clásica en la que tenemos una cabecera y un footer comunes a ambas vistas. En la parte central encontramos el body, esta parte si cambia en función de la vista.

4.2.3.1. Diseño de la Vista Escribir Texto

Ilustración 32: Wireframe de la Vista Escribir Texto

- **Título:** Identificador de la página.
- **Botón 1:** Selector del tipo de recurso a clasificar. Según el seleccionado se cambiará de una vista a otra.
- **Texto 1:** Identificador de la vista.
- **Texto 2:** Espacio para mostrar la información resultante de una clasificación o mensajes de error, al acceder a la vista esta zona estará oculta.
- **Botón 2:** Botón para seleccionar el modelo clasificador.
- **Botón 3:** Botón para clasificar el texto introducido.
- **Texto 3:** Información adicional de la página situada en el pie de página.

4.2.3.2 Diseño de la Vista Subir Archivo

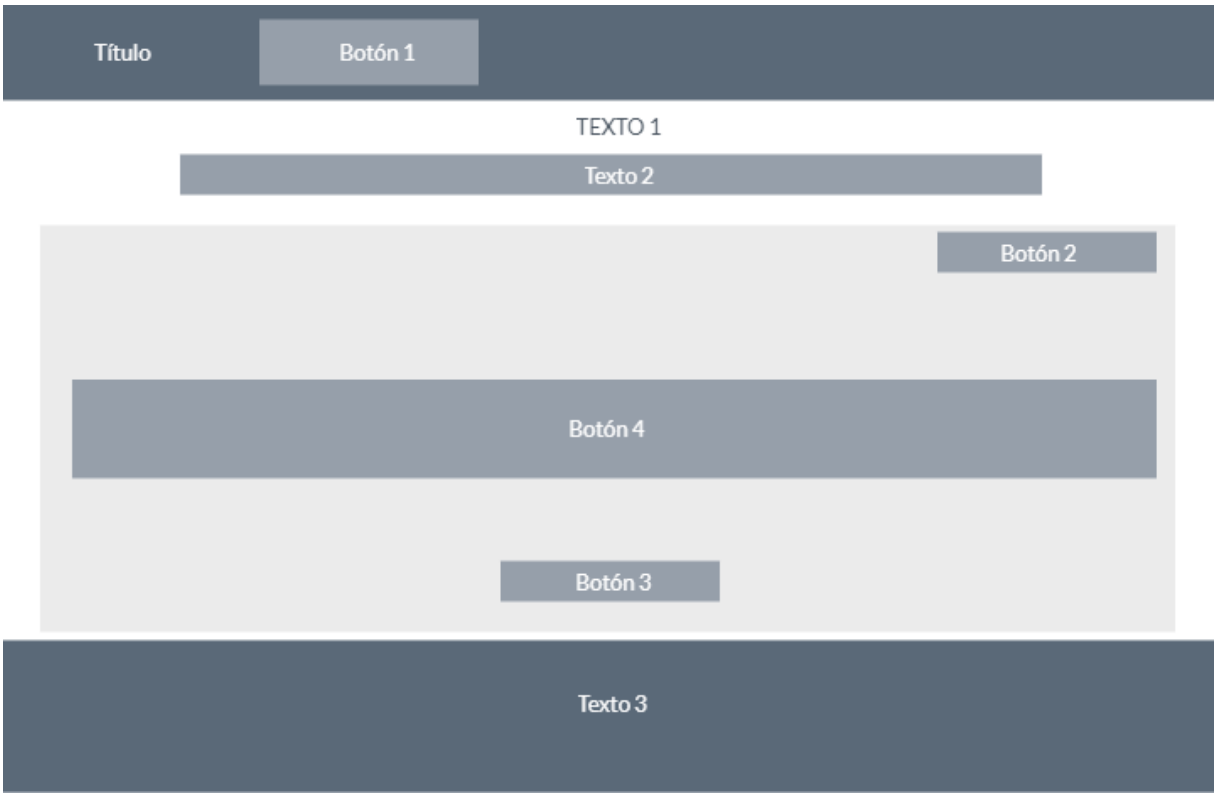

Ilustración 33: Wireframe de la Vista Subir Archivo

- **Título:** Identificador de la página.
- **Botón 1:** Selector del tipo de recurso a clasificar. Según el seleccionado se cambiará de una vista a otra.
- **Texto 1:** Identificador de la vista.
- **Texto 2:** Espacio para mostrar la información resultante de una clasificación o mensajes de error, al acceder a la vista esta zona estará oculta.
- **Botón 2:** Botón para seleccionar el modelo clasificador.
- **Botón 3:** Botón para clasificar el texto introducido.
- **Texto 3:** Información adicional de la página situada en el pie de página.
- **Botón 4:** Botón para subir un archivo en formato *.docx.*

5. DESARROLLO E IMPLEMENTACIÓN DE LA APLICACIÓN

En este capítulo se detalla el trabajo realizado en la creación del sistema clasificador. Se muestra información sobre las funciones implementadas, particularidades del sistema, librerías python, equipo usado y finalmente, el proyecto resultante.

5.1. Equipo Usado.

- Procesador: Intel i7-3630QM CPU @ 2.40GHz 2.40 GHz
- RAM: 8 GB

- Sistema Operativo: Windows 10 Home

5.2. Instalación de PyCharm y Python.

En primer lugar, se procedió a descargar el IDE PyCharm 2022.3.2 desde el portal oficial de JetBrains.

Posteriormente se instaló Python en su versión 3.9 de 64 bits en el equipo.

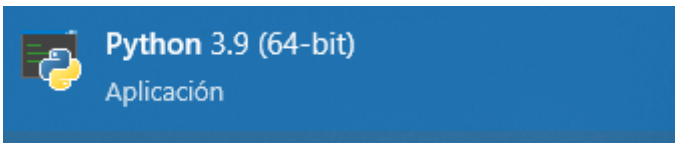

Ilustración 34: Instalación de Python

5.3. Librerías de Python Utilizadas

5.3.1. Spacy

Esta librería se ha usado para el procesamiento de los textos de cara a alimentar los modelos que no son BERT.

5.3.2. Transformers

Librería para la creación del modelo BERT, su respectivo clasificador, tokenizador, pipeline, trainer y training arguments.

5.3.3. Os

Con esta biblioteca se accedía a los directorios del equipo y su respectivo contenido mediante una ruta por parámetro.

5.3.4. Flask

Para la parte de la implementación de la página web se ha utilizado la librería Flask tal y como se ha explicado anteriormente.

5.3.5. Glob

Utilizada para la lectura de archivos de un determinado tipo en un directorio. En concreto la lectura de los archivos *.docx* del corpus de denuncias policiales.

5.3.6. Doc2txt

Librería de gran utilidad para transformar el contenido de un documento *.docx* en un texto en formato cadena, facilitando así el tratamiento de los archivos del corpus.

5.3.7. Sk-learn

Utilizada principalmente para la validación cruzada y evaluación de los métodos. Es decir, la generación de las métricas, reportes de métricas y división del conjunto de datos mediante la función k-fold.

5.3.8. Itertools

Esta librería se ha usado para cortar listas en una sublista de un tamaño definido por parámetro. Utilizada en la función de etiquetado de textos.

5.3.9. Numpy

Librería utilizada para el tratamiento de predicciones de los modelos transformers y su respectiva conversión a vectores binarios.

5.3.10. Pandas

Con esta librería se facilitaba el proceso de creación de estructuras de datos indexadas con fácil conversión a csv.

5.3.11. Dataset

Como su propio nombre indica, utilizada para transformar una estructura al formato Dataset. Formato el cual era requerido en algunos pasos del entrenamiento el modelo BERT.

5.3.12. Sys

Utilizada para la lectura y escritura de archivos mediante el comando open. Por ejemplo, la escritura de una línea de texto en un archivo csv.

5.3.13. Csv

Esta librería se ha usado para convertir una conjunto datos en un archivo csv sin necesidad de tener que crear una función iterativa con la librería Sys.

5.3.14. Time

Con esta librería se calculaba el tiempo de ejecución de la validación cruzada.

5.3.15. Shutil

Usada para eliminar directorios o contenido de estos se ha utilizado Shutil. Por ejemplo, para evitar el almacenamiento excesivo en el equipo de modelos BERT se hacía uso de esta para el borrado de la respectiva carpeta de modelos.

5.3.16. Random

Tal y como su nombre indica, es utilizada para la generación de números aleatorios.

5.3.17. Re

Esta librería ha sido utilizada en *procesamiento3()* para detectar fragmentos de textos que empiezan por una palabra y acaban en otra.

5.3.18. Joblib

Mediante las respectivas funciones de carga y almacenamiento de esta librería se ha implementado la persistencia de los modelos que no son BERT y sus correspondientes vectorizadores de cara al acceso desde la aplicación web.

5.4. Particularidades del sistema clasificador.

5.4.1. Corpus

Contamos con un Corpus de archivos de denuncias policiales proporcionado por el Ministerio de Interior. Disponemos de 110 atestados correspondientes a denuncias de delitos de Odio y 80 a delitos de No Odio. Se encuentran almacenadas en carpetas diferentes por tipos y en formato .docx. Este corpus se encuentra en la carpeta */TFG/Corpus.* Al no disponer de permiso este corpus no será incluido en la entrega del proyecto.

5.4.1. Pre-procesamientos

Los modelos a entrenar serán alimentados con este corpus etiquetado. Sin embargo, como paso previo, se implementará una serie de funciones a través de las cuales se tratarán y/o reducirán su número de instancias de palabras. Todo ello de cara a medir y entender los resultados con diferentes procesamientos. A continuación, una breve descripción de cada procesamiento. Se puede encontrar más información y un análisis más detallado en el punto 7.

5.4.1.1 Procesamiento 1

- Quita los espacios vacíos del formato .docx.

5.4.1.2 Procesamiento 2

- Quita los espacios vacíos del formato .docx.
- Elimina los signos de puntuación.
- No se tienen en cuenta los carácteres que no son alfabéticos:
    - Cabecera con el nombre del documento.
    - Etiquetas de anonimizado.

5.4.1.3 Procesamiento 3

- Quita los espacios vacíos del formato .docx.
- No se tienen en cuenta los carácteres que no son alfabéticos:

    - Cabecera con el nombre del documento.
    - Etiquetas de anonimizado.
- Elimina los signos de puntuación.
- Elimina frase común al tipo de denuncia:
    - NO ODIO: No tiene en cuenta las palabras desde “Atestado” hasta “Dependencia”.
    - ODIO: No tiene en cuenta las palabras desde “Instructor” hasta “ocurridos”.

5.4.1.4 Procesamiento 4a

- Quita los espacios vacíos del formato .docx.
- No se tienen en cuenta los carácteres que no son alfabéticos:
    - Cabecera con el nombre del documento.
    - Etiquetas de anonimizado.
- Elimina los signos de puntuación.
- Elimina las siguientes palabras de cada tipo de denuncia, sin tener en cuenta stop words:
    - Palabras comunes de cada clase, es decir, palabras presentes en todos los documentos de una misma clase.
    - Palabras más repetidas de cada clase.
    - Palabras exclusivas de cada clase, es decir, palabras que aparecen en todos los documentos de una misma clase y no aparecen en el otro tipo de denuncia.

5.4.1.5 Procesamiento 4b

- Se trata del mismo funcionamiento que el procesamiento 4 pero teniendo en cuenta stop words en la eliminación de palabras comunes repetidas o exclusivas.

5.4.2. Modelos

De entre todos los modelos estudiados anteriormente se han elegido algunos para realizar los entrenamientos, pruebas y correspondiente evaluación de métricas.

Como pilar fundamental del proyecto encontramos los siguientes modelos BERT: **Distilbert, Beto, María, Bertín y Multilingual-bert**.

Por otro lado, para obtener más variedad de resultados y clarificar su veracidad se implementará este entrenamiento y evaluación con modelos que no son BERT. Todo ello para comprobar si se estaba sobreentrenando el bias o eran incorrectas y/o demasiado altas las métricas recibidas. Los modelos son los siguientes: **SVM y Naive Bayes**.

5.5. Implementación del Modelo.

Durante la fase de implementación también se utilizó la metodología incremental detallada en la planificación del proyecto. En primer lugar se desarrollaban pequeños prototipos con funcionalidad completa y con el paso de

tiempo se iban mejorando y haciendo más complejos con todas las novedades aprendidas.

Para mayor exhaustividad en la representación de la implementación se procederá, a continuación, a explicar mediante una breve descripción de cada función implementada junto con su representación en pseudocódigo.

Las funciones de muestra de información que se limita a hacer un print() de las variables no será reflejadas en pseudocódigo, ya que carecen de interés de cara a la explicación de la implementación.

5.5.1. Clase Configuración

5.5.1.1. lecturaParámetros

Esta función se encarga de recibir la información del fichero de parámetros y la almacena en las variables de la clase Configuración.

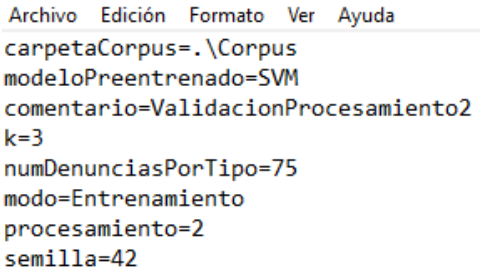

Ilustración 35: Fichero de Configuración

Se realiza un bucle por cada fila del fichero de parámetros. El contenido se separa mediante un split tomando como punto medio el signo “=”. La primera parte de esta separación será usada para identificar la variable del fichero de parámetros y la segunda parte será identificada como el contenido a guardar en dicha variable.

```
/**Configuración Lectura Parámetros**/
función lecturaParámetros()
    for linea in open(Parametros.txt)
    contenido = linea.split(=)
    if contenido[0] == "carpetaCorpus"
       self.carpetaCorpus=contenido[1]
    fin if
    if contenido[0] == "modeloPreentrenado"
       self.modeloPreentrenado=contenido[1]
    fin if
    if contenido[0] == "nombre"
       self.nombre=contenido[1]
    fin if
    if contenido[0] == "k"
       self.k=contenido[1]
    fin if
    if contenido[0] == "numDenunciasPorTipo"
       self.numDenunciasPorTipo=contenido[1]
    fin if
    if contenido[0] == "modo"
       self.modo=contenido[1]
    fin if
    if contenido[0] == "procesamiento"
       self.procesamiento=contenido[1]
    fin if
    if contenido[0] == "semilla"
       self.semilla=contenido[1]
    fin if
Fin función
```

Ilustración 36: Pseudocódigo de lecturaParámetros()

5.5.2. Clase LecturaArchivos

5.5.2.1. Cargar()

En esta función, en primer lugar, nos situamos en la carpeta del corpus. Posteriormente, por cada carpeta contenida en esta se abrirán todos los archivos docx con la librería glob y serán convertidos en txt con la librería docx2txt.

```
/**Lectura Archivos cargar()**/
funcion cargar()
    directorio=configuracion.getCarpetaCorpus()
    for carpeta in directorio
        for archivo in glob.buscarArchivos(.docx)
            archivo=open(archivo)
            texto=docx2txt.procesar(archivo)
            if texto!=""
                mapaCarpetas[carpeta].append(texto)
            fin if
    fin for
Fin funcion
```

Ilustración 37: Pseudocódigo de cargar()

Finalmente, son almacenados en un mapa que contiene una lista de textos por cada carpeta.

5.5.3. Clase Modelo

5.5.3.1. FuncionTokenizadora()

```
/**Modelo funcionTokenizadora()**/
función funcionTokenizadora()
    tokenizer=Autotokenizer.fromPretrained(modelo)
    return tokenizer(textos, batched=true)
Fin función
```

Ilustración 38: Pseudocódigo de funcionTokenizadora()

Se trata de la función tokenizadora para los modelos BERT, se define el tokenizador en función del modelo a usar y se establece la estructura de la tokenización.

5.5.3.2. Etiquetar()

```
función etiquetar(numeroArchivos)
    if numeroArchivos == -1
        for archivo in self.delitosOdio
            textoEtiquetado[texto]=archivo
            etiqueta=0
            listaTextos.append(textoEtiquetado)
        Fin for
        for archivo in self.delitosNoOdio
            textoEtiquetado[texto]=archivo
            etiqueta=1
            listaTextos.append(textoEtiquetado)
        Fin for
    else
        for archivo in self.delitosOdio.cortarHasta(numeroArchivos)
            textoEtiquetado[texto]=archivo
            etiqueta=0
            listaTextos.append(textoEtiquetado)
        Fin for
        for archivo in self.delitosNoOdio.cortarHasta(numeroArchivos)
            textoEtiquetado[texto]=archivo
            etiqueta=1
            listaTextos.append(textoEtiquetado)
        Fin for
    Fin if
    return listaTextos
Fin función
```

Ilustración 39: Pseudocódigo de etiquetar()

Esta función es usada para asignar una etiqueta al conjunto de textos extraídos de la carpeta del corpus y guardarlos en una lista. Según el vector del que procedan, se les asignará su correspondiente etiqueta. En este caso al tratarse de un clasificador binario representaremos con un 1 la denuncia de No Odio y con un 0 la denuncia de Odio.

Cuenta con dos funcionamientos según el valor recibido por parámetro. Si el número de textos a etiquetar está definido como -1 se etiquetará todo el conjunto de

texto. Si está definido con cualquier otro valor n, se etiquetará ese número de textos n.

5.5.3.3. MuestraMericas()

```
función muestrametricas(etiquetas,predicciones)

    recall = sklearn.recall(etiquetas,predicciones,average=macro)
    precision = sklearn.precision(etiquetas,predictions,average=macro)
    flscore = sklearn.flscore(etiquetas,predictions,average=macro)

    reporte=mostrarMetricas(labels,predicions)

    mostrar(recall, accuracy, precision,flscore,reporte)
    return reporte

Fin función
```

Ilustración 40: Pseudocódigo de muestraMetricas()

En esté método se usa la biblioteca sklearn para calcular las métricas de validación del modelo. Se recibe un conjunto de etiquetas reales y un conjunto de etiquetas predichas para una serie de textos. Estas variables recibidas se traspasan a sus respectivas funciones de sklearn (recall, precision, score, generador de reportes) y son devueltas mediante un return.

5.5.3.4. MetricasComputo()

```
función metricasComputo(evaluacion)

    predictions=numpy.ConvertitirAEtiquetas(evaluacion.predicciones;
    labels=evaluacion.etiquetas

    recall = sklearn.recall(labels,predictions,average=macro)
    accuracy = sklearn.accuracy(labels,predictions)
    precision = sklearn.precision(labels,predictions,average=macro)
    flscore = sklearn.flscore(labels,predictions,average=macro)

    reporte=sklearn.reporte(labels,predictions)

    return {recall, accuracy, precision,flscore,reporte}

Fin función
```

Ilustración 41: Pseudocódigo de metricasComputo()

En métricasComputo() se define una función de cómputo que es pasada en el entrenamiento del modelo transformers. Es usada para validar el modelo en medio de la ejecución en las diferentes fases. Aunque es el mismo proceder que en el cálculo de métricas anteriormente desarrollado, es bastante orientativa para ojear las métricas en mitad del entrenamiento. Sin embargo, no es representativo del rendimiento del modelo.

5.5.3.5. Entrenar()

```
función entrenar(numArchivos)
    listaTextos=etiquetar(numArchivos)
    if self.configuracion.getModeloPreentrenado in self.modelosBERT

        dataSetPanda=pd.convetirADataFrame(listaTextos)
        dataSet=dataset.convertirADataset(dataSetPanda)
        dataSetMapTokenizado=dataSet.map(funcionTokenizadora,batched=true)
        self.entrenarBERT(dataSetMapTokenizado,dataSetMapTokenizado,metricasComputo
    Fin if

    if self.configuracion.getModeloPreentrenado in self.modelosBERT
        self.entrenarNoBERT(listaTextos)
    Fin if

Fin función
```

Ilustración 42: Pseudocódigo de entrenar()

En primer lugar, se invoca a la función de etiquetado. Posteriormente, esta función consta de dos condiciones.

Si recibimos por parámetro un modelo BERT se realiza una transformación a Panda, DataSet y posteriormente se convierte en un Map tokenizado con el tokenizador BERT. Seguidamente, se llama a la función de entrenamiento con este conjunto tokenizado tanto como conjunto de validación como de entrenamiento, ya que se trata de un entrenamiento y no una validación, junto con la función de métricas de cómputo.

5.5.3.6. EntrenarBERT()

```
función ejecutar(numArchivos)
    id2label = {0: "ODIO", 1: "NO ODIO"}
    label2id = {"ODIO": 0, "NO ODIO": 1}
    model=transformers.AutoModelForSecuenceClassification(configuracion.modeloPreentrenado,num_labels=2,id2label,label2id)
    transformers.dataCollatorWithPadding(self.tokenizer)
    parametrosDeEntrenamient(.....)
    trainer=trainsformers.trainer(
        model=model,
        args=parametrosDeEntrenamient,
        train_dataset=datasetTokenizadoTrain,
        eval_dataset=datasetTokenizadoVal,
        tokenizer=self.tokenizer,
        data_collator=data_collator,
        compute_metrics=compute_metrics,
    )

    trainer.train()
Fin función
```

Ilustración 43: Pseudocódigo de entrenarBERT()

En esta función se define a qué etiqueta pertenece cada índice mediante el parámetro id2label. Posteriormente se crea el modelo mediante la biblioteca transformers estableciendo el número de etiquetas y se definen los parámetros de entrenamiento y los parámetros del trainer (Huggin Face, s.ff.):

Parámetros de entrenamiento:

- **evaluation-strategy:** Estrategia de evaluación, puede ser por épocas.
- **save_strategy:** Estrategia de guardado, puede ser por épocas.
- **learning_rate:** Ratio de entrenamiento del modelo.
- **per_device_train_batch_size:** Tamaño de los lotes del conjunto de entrenamiento.
- **per_devive_val_barch_size:** Tamaño de los lotes del conjunto de validación.
- **num_train_epochs:** Número de épocas del entrenamiento, veces que se entrena.
- **weight_decay:** Caída o deterioro de los pesos.
- **load_best_model_at_end:** Variable que indica si se carga el mejor modelo al final.
- **push_to_hub:** Variable que indica si se sube al sitio web el modelo.
- **output_dir:** Directorio en el que guardar el modelo.

Parámetros del trainer (Huggin Face, s.ff.):

- **model:** Nombre del modelo con el que entrenar.
- **args:** Parámetros de entrenamiento.
- **train_dataset:** Conjunto de datos de entrenamiento.
- **val_dataset:** Conjunto de datos de validación.
- **tokenizer:** Tokenizador.
- **data_collator:** Colector de datos.
- **compute_metrics:** Instancia de las variables de cómputo.

5.4.3.6. EntrenarNoBERT()

```
función entrenarNoBERT(listaTextos)
    def tokenizar(texto): //Función de tokenizado

        texto = Spacy.nlp(texto)
        for token in texto
            if token.text.isalpha() and not token.is_stop
                tokens.append(token.lemma_.lower())
        textoProcesado = unirEnUnaCadena(tokens)
        return textoProcesado

    textos =listaTextos["text"]
    etiquetas=listaTextos["label"]

    if configuracion.getModeloPreentrenado=="SVM"
        vectorizador=Sklearn.TFIDF
        modelo=Sklearn.SVM
    Fin if

    if configuracion.getModeloPreentrenado=="Naive"
        vectorizador=Sklearn.TFIDF
        modelo=Sklearn.NaiveBayes
    Fin if

    modelo.fit(vectorizador.fit(textos), etiquetas)

    nombreCarpeta=self.configuracion.getModeloPreentrenado+"-"+self.configuracion.getComentario
    os.mkdir(nombreCarpeta)
    os.chdir(nombreCarpeta)
    Joblib.dump(model, 'Modelo.joblib')
    Joblib.dump(vectorizer,'Vectorizador.joblib')

Fin función
```

Ilustración 44: Pseudocódigo de entrenarNoBERT()

Esta función es usada para entrenar modelos que no son BERT.

En primera instancia, se define una función de tokenización aprendida en la asignatura Procesamiento del Lenguaje Natural. En este método se invoca a la función nlp de Spacy para dividir el texto en un conjunto de tokens. Seguidamente, de entre estos tokens sólo nos quedaremos en formato minúscula con los que son alfabéticos ni stop words.

Por otro lado, establecemos un vector de textos y otro de etiquetas extraídos del mapa listaTextos. Seguidamente, contamos con una secuencia de condiciones para identificar si estamos ante un modelo SVM o uno Naive Bayes. Para estas dos condiciones, definimos un vectorizador TFIDF y su respectiva invocación desde SkLeran.

Establecemos un ajuste del modelo con el fit de los textos en el vectorizador y el conjunto de etiquetas. Finalmente, guardamos tanto el modelo entrenado como el vectorizador en una carpeta con la librería *JobLib* para poder acceder a ella desde la web.

5.5.3.7. CalculaMediaMetricas()

```
función calculaMediaMetricas(numArchivos)
    for metricaPanda in self.metricas()
        i=0
        while i<5
            j=0
            while j<5
                vector.append(metricaPanda.obtenerPosicion[i,j]
                j++
            i++
        fin while
        vectorMetricas.append(vector)
    fin for
    for vector in vectorMetricas
        for posicon in vector:
            vSum[i]=vSum[i]+elemento[i]
    fin for

    for suma in vectorSumas
        if vSum[i]==0
            vSum[i]==0
        else
            vSum[i]=vSum[i]/k
        fin if
        i++
    fin for
    datos(metricaConLosDatosDE(vSum)
    metrica=pandas.convertirADataFrame(datos)
    metrica.toCSV
Fin función
```

Ilustración 45: Pseudocódigo de calculaMediaMetricas()

Función usada para calcular la media de todas las métricas obtenidas en una validación cruzada.

En primer lugar, nos dedicamos a iterar todo el panda de cada métrica y transformarlo en un vector para guardar esos vectores extraídos en otro vector de vectores.

Posteriormente, por cada vector de métricas iremos sumando cada posición y lo almacenaremos en otro vector con la suma de cada posición.

Finalmente, por cada posición del vector de sumas se realiza la media dividiéndola entre la *k* de la validación cruzada.

5.5.3.8. ExtraeMetricas()

```
función extraeMetricas(reporte)

    lineas = reporte.split('\n')
    linea1 = lineas[2]
    linea2 = lineas[3]
    linea3 = lineas[5]
    linea4 = lineas[6]
    linea5 = lineas[7]

    vectorLinea1 = linea1.split()
    vectorLinea2 = linea2.split()
    vectorLinea3 = linea3.split()
    vectorLinea4 = linea4.split()
    vectorLinea5 = linea5.split()

    datos('precision': [float(vectorLineal[1])..............)

    self.metricas.append(pandas.ConvertirADataFrame(datos,índices))
Fin función
```

Ilustración 46: Pseudocódigo de extraeMetricas()

Esta función es usada principalmente para convertir un reporte de clasificación de sklearn en un panda indexado para tener acceso al valor de cada métrica. Se corta el reporte por líneas y posteriormente a esas líneas se les realiza un split por espacios. Este split será almacenado en un vector para su respectiva línea.

Como paso final, se devolverá un panda creado por índices con los respectivos valores extraídos anteriormente.

5.5.3.9. ValidaciónCruzadaBERT()

```
función crossValidationBERT(numTextos,k)
    listaTextos=etiquetar(numArchivos)
    inicio=time.time()
    for indices_train, indicesl_val in sklearn.StratifiedKFold(k,shuffle=True)
        for i in indicesTrain
            train.append(listaTextos[i])
        fin for
        for i in indices_val
            val.append(listaTextos[i]
        fin for

        datasetDatasetTrain= Dataset.convertirADataset(pandas.convetirADataFrame(train))
        datasetDatasetVal = Dataset.convertirADataset(pandas.convetirADataFrame(val))

        datasetTokenizadoTrain = datasetDatasetTrain.map(self.funcionTokenizadora, batched=True)
        datasetTokenizadoVal = datasetDatasetVal.map(self.funcionTokenizadora, batched=True)

        model = self.entrenar(datasetTokenizadoTrain, datasetTokenizadoVal, self.metricasComputo)
        clasificador = pipeline("text-classification", truncation=True, model=model, tokenizer=self.tokenizer,
                                batch_size=16)

        while i < listaTextos.size():
            if i in indices_val:
                etiquetas.append(listaTextos[i]['label'])
                clasificacion = classifier(listaTextos[i]['text'])

                if str(clasificacion)[12:16] == 'ODIO':
                    predicciones.append(0)
                else:
                    predicciones.append(1)
            i++
            reporte=sklearn.classificationReport
            extraeMetricas(reporte)
    fin for
    fin=time.time()
    self.tiempoValidacionCruzada=fin-inicio
    self.calculaMediaMetricas(numDivisiones)
Fin función
```

Ilustración 47: Pseudocódigo de validaciónCruzadaBERT()

Esta función es usada para la validación cruzada de los modelos BERT. Mediante el método de sklearn *StratifiedKFlod* se divide el corpus *k* conjuntos. *k-1* de esos conjuntos son utilizados para el entrenamiento y el *k* restante para la validación. Cabe destacar que esta función realiza la división manteniendo el porcentaje de etiquetas de cada tipo.

Como resultado de la función anteriormente explicada, obtenemos dos vectores de índices, uno para el conjunto de validación y otro para el conjunto de entrenamiento. En total se realizarán *k* iteraciones para validar cada conjunto train con los restantes de entrenamiento. Por cada iteración, crearemos dos listas con sus respectivos textos y serán transformados a un mapa tokenizado siguiendo el mismo procedimiento que en la función ejecutar().

Posteriormente, se crea el modelo BERT, se llamará a la función entrenar(), y una vez entrenado, se creará el clasificador con ese modelo entrenado.

Por último, por cada texto del conjunto de validación se llamará al clasificador y se procesará la predicción obtenida para convertirla en un vector binario. Con el conjunto de etiquetas predichas y las reales se llama al *classificationReport* de *Sk-learn*. El reporte obtenido será almacenado mediante la función *extraeMétricas()*.

Una vez completadas todas las iteraciones se invoca a la función del cálculo de medias y se computa el tiempo que ha tardado la validación cruzada de este modelo.

5.5.3.10. Preprocesar para no BERT()

```
función procesar(texto)
    texto=spacy.lnp(texto)
    for token in texto
        if token.text.esalfabético and not token.esStopWord
            listaTokens.append(token.lema.minúscula))
    textoProcesado="".join(tokens)
    return textoProcesado
    fin for
fin función
```

Ilustración 48: Pseudocódigo de preprocesar()

La función procesar es utilizada para tokenizar los textos con la librería *Spacy* con el fin de alimentar los modelos No Bert. Se llama a la función *nlp()* y por cada token generado, se guardan en la lista convertidos completamente a minúscula. Siempre aquellos que sean alfabéticos y ni son stop words.

Como último paso, se convierte la lista de tokens en una cadena de texto separada por espacios entre cada token.

5.5.3.11. ValidaciónCruzadaBERT()

```
función crossValidationOtrosModelos(numTextos,k)
    listaTextos=etiquetar(numArchivos)
    inicio=time.time()
    for indices_train, indicesl_val in sklearn.StratifiedKFold(k,shuffle=True)
        for i in indicesTrain
            train.append(listaTextos[i])
        fin for
        for i in indices_val
            val.append(listaTextos[i]
        fin for

        for i in indices_train:
            trainFinalTexto.append(procesar(listaTextos[i]['text']))
            trainFinalEtiqueta.append(listaTextos[i]['label'])
        fin for

        vectorizador=sklearn.tfidfvectorizer
        X = vectorizador.fit(trainFinalTexto)
        y=trainFinalEtiqueta
        model = svm.SVC()
        model.fit(X, y)

        for elemento in valFinalTexto:
            Y = vectorizer.transform(elemento)

            prediction = model.predict(Y)
            predicciones.append(prediction[0])
        fin for
        reporte = sklearn.classification_report(valFinalEtiqueta, predicciones)
        self.extraeMetricas(reporte)
    fin for
    fin=time.time()
    self.tiempoValidacionCruzada=fin-inicio
    self.calculaMediaMetricas(numDivisiones)
Fin función
```

Ilustración 49: Pseudocódigo de validacionCruzadaBERT()

Esta función es usada para la validación cruzada de los modelos que no son BERT. Al igual que en la validación cruzada de modelos BERT, mediante el método de *Sklearn StratifiedKFlod* se divide el corpus *k* conjuntos. *k-1* de esos conjuntos son utilizados para el entrenamiento y el *k* restante para la validación.

También, como resultado de la función obtenemos dos vectores de índices, uno para el conjunto de validación y otro para el conjunto de entrenamiento. En total se realizarán *k* iteraciones de cara a validar cada conjunto train con los restantes de entrenamiento. Por cada iteración, crearemos dos listas con sus respectivos textos y serán transformados a un mapa tokenizado siguiendo el mismo procedimiento que en la función ejecutar().

Posteriormente, se crea el modelo recibido por parámetro y el respectivo vectorizador. Se hará un fit del vectorizador con el conjunto de textos y este junto con la lista de etiquetas será recibido como parámetro en el fit del modelo.

Por último, al igual que en la validación de los modelos BERT, por cada texto del conjunto de validación se llamará al modelo.predecir etiquetas.

Con el conjunto de etiquetas predichas y las reales se llama al Classification report de Sk-learn. El reporte obtenido será almacenado mediante la función *extraeMétricas*.

Una vez completadas todas las iteraciones se invoca a la función del cálculo de medias y se computará el tiempo que ha tardado la validación cruzada de este modelo.

5.5.3.11. CrearCarpetaValidacionCruzada()

```
función crearCarpetasValidacionCruzada(train,val)
    os.chdir("./Corpus")
    carpetas=Os.listarDirectorio()
    for indice in train
       if indice < self.numNoOdio:
            Os.chdir(carpetas[0])
            listaTextos=os.listdir()
            carpetaInicial = ..\Corpus\NOODIO'
            carpetaFinal = ..\PruebaEntrenamiento\NOODIO'
            Shutil.copiarArchivo(carpetaInicial+"\"+listaTextos[indice], carpetaFinal+"\"+listaTextos[indice])

        else:
            Os.chdir(carpetas[1])
            listaTextos=os.listdir()
            carpetaInicial = ..\Corpus\ODIO'
            carpetaFinal = ..\PruebaEntrenamiento\ODIO'
            Shutil.copiarArchivo(carpetaInicial+"\"+listaTextos[indice-self.NumNoOdio]), carpetaFinal+"\"+listaTextos[indice-self.NumNoOdio])
        Fin if
    Fin for

    for indice in val
       if indice < self.numNoOdio:
            Os.chdir(carpetas[0])
            listaTextos=os.listdir()
            carpetaInicial = ..\Corpus\NOODIO'
            carpetaFinal = ..\PruebaValidacion\NOODIO'
            Shutil.copiarArchivo(carpetaInicial+"\"+listaTextos[indice]), carpetaFinal+"\"+listaTextos[indice])

        else:
            Os.chdir(carpetas[1])
            listaTextos=os.listdir()
            carpetaInicial = ..\Corpus\ODIO'
            carpetaFinal = ..\PruebaValidacion\ODIO'
            Shutil.copiarArchivo(carpetaInicial+"\"+listaTextos[indice-self.NumNoOdio], carpetaFinal+"\"+listaTextos[indice-self.NumNoOdio])
        Fin if
    Fin for
Fin Función
```

Ilustración 50: Pseudocódigo de crearCarpetaValidacionCruzada()

Esta función recibe un conjunto de índices que han sido utilizados en una validación cruzada, índices de entrenamiento y de validación. En esta función primero se listan todos los archivos del directorio *.\Corpus*. Posteriormente, contamos con dos bucles.

Por cada índice del conjunto train. Si el índice es menor del número de delitos de No Odio, buscaremos el delito por su índice en la carpeta de delitos No Odio y la copiaremos con la función Shutil en la carpeta final correspondiente.

Si el índice es mayor que el número de delitos de Odio, el proceder será el mismo pero buscando el índice del delito de la carpeta de delitos de Odio. Este índice será buscado como *índice - numDelitosNoOdio*, ya que estamos buscando en la carpeta de Odio por lo que tenemos que descartar los índices de los delitos que no son de odio.

Por cada índice del conjunto *val*, seguiremos el mismo procedimiento descrito anteriormente, pero cambiando la carpeta de destino de la copia de los archivos.

5.5.4. Clase ProcesaTexto()

5.5.4.1. Procesaminento1()

```
funcion procesar1()
    i = 0
    for delito in delitosODIO
        palabras = delito.split()
        for palabra in palabras
            listaAlfa.append(palabra)
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        self.delitosodio[i] = textoAlfa
        i = i + 1
    Fin for
    i = 0
    for delito in delitosNoOdio
        for letra in delito
            lista.append(letra)
        Fin for
        texto = ''.join(lista)
        palabras = texto.split()
        for palabra in palabras
            listaAlfa.append(palabra)
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        self.delitosnodio[i] = textoAlfa
        i = i + 1
    Fin for
Fin función
```

Ilustración 51: Pseudocódigo de procesamiento1()

Encontramos dos bucles, el primero para procesar los textos de ODIO y el segundo para procesar los textos de No ODIO. En ambos, se realiza un split por espacios de la cadena de texto donde posteriormente, son almacenados en una lista separada por un espacio cada palabra. Con ello, conseguimos eliminar los espacios excesivos generados por el formato *docx*. Cabe destacar que este procesamiento se utiliza por pura visualización.

5.5.4.2. Procesamiento2()

```
funcion procesar2()
    i = 0
    for delito in delitosODIO
        for letra in delito
            lista.append(letra)
        Fin for
        texto = ''.join(lista)
        palabras = texto.split()
        for palabra in palabras
            if not palabra.isupper()
                if palabra.isalpha() or palabra.isnumeric()
                    listaAlfa.append(palabra)
                Fin if
            Fin if
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        self.delitosodio[i] = textoAlfa
        i = i + 1
    Fin for
    i = 0
    for delito in delitosNoOdio
        for letra in delito
            lista.append(letra)
        Fin for
        texto = ''.join(lista)
        palabras = texto.split()
        for palabra in palabras
            if not palabra.isupper()
                if palabra.isalpha() or palabra.isnumeric()
                    listaAlfa.append(palabra)
                Fin if
            Fin if
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        self.delitosnodio[i] = textoAlfa
        i = i + 1
    Fin for
Fin función
```

Ilustración 52: Pseudocódigo de procesamiento2()

En primer lugar y para ambos conjuntos de denuncias, se recorren todos los carácteres de la cadena y sólo nos quedamos con los que no son signos de puntuación. Posteriormente, por cada palabra en la cadena procesada restante se almacenan en un vector aquellas que no están completamente en mayúscula y son alfabéticas o numéricas.

Para finalizar, esta lista de palabras obtenida será transformada a una cadena donde cada palabra estará separada por un espacio.

5.5.4.3. Procesamiento3()

```
funcion procesar3()
    i = 0
    for delito in delitosODIO:
        for letra in delito
            lista.append(letra)
        Fin for
        texto = ''.join(lista)
        palabras = texto.split()
        for palabra in palabras
            if not palabra.isupper()
                if palabra.isalpha() or palabra.isnumeric()
                    listaAlfa.append(palabra)
                Fin if
            Fin if
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        patron = r"Instructor\s(.*?)\socurridos"
        texto_modificado = re.sub(patron, '', textoAlfa)
        self.delitosodio[i] = texto_modificado
        i = i + 1
    Fin for
    i = 0

    for delito in delitosNoOdio
        for letra in delito
            lista.append(letra)
        Fin for
        texto = ''.join(lista)
        palabras = texto.split()
        for palabra in palabras
            if not palabra.isupper()
                if palabra.isalpha() or palabra.isnumeric()
                    listaAlfa.append(palabra)
                Fin if
            Fin if
        Fin for
        textoAlfa = ' '.join(listaAlfa)
        patron = r"Atestado\s(.*?)\Dependencia"
        texto_modificado = re.sub(patron, '', textoAlfa)
        self.delitosnodio[i] = texto_modificado
        i = i + 1
    Fin for
Fin función
```

Ilustración 53: Pseudocódigo de procesamiento3()

El procedimiento inicial es similar a la función anterior. En primer lugar para ambos conjuntos de denuncias, se recorren todos los carácteres de la cadena y sólo nos quedamos con los que no son signos de puntuación.

Posteriormente, por cada palabra en la cadena procesada restante se almacenan en un vector aquellas que no están completamente en mayúscula y son alfabéticas o numéricas.

En la parte final, se añade una capa extra de procesamiento. Mediante la librería de expresiones regulares “Re” generamos una expresión regular para

representar a todas aquellas frases que empiezan y acaban por una palabra. Todo ello con el objetivo de eliminar las frases repetitivas iniciales propias del formato de denuncia según cada tipología:

- Denuncia de Odio: Se eliminan todas las palabras entre “Instructor” y “ocurridos”.
- Denuncia de No Odio: Se eliminan todas las palabras entre “Atestado” y “Dependencia”.

5.5.4.4. Procesamiento4()

```
def procesamiento4(tenerEnCuentaStopWords)
    procesamiento2()
    procesamiento4CalcularPalabras(nComunes=500, tenerEnCuentaStopWords=tenerEnCuentaStopWords)
    procesamiento4AplicarPalabras(self.delitosnodio)
    procesamiento4AplicarPalabras(self.delitosodio)
Fin funcion
```

Ilustración 54: Pseudocódigo de procesamiento4()

Esta función recibe un parámetro que indica si vamos a tener en cuenta las stopwords en el eliminado de palabras.

En primer lugar, realiza un tratamiento inicial con el *procesamiento2()*. Posteriormente, invoca la función que calculará las palabras a eliminar y aplicará ese borrado de palabras sobre los dos vectores de denuncias mediante el llamado a sus respectivas funciones.

5.5.4.5. Procesamiento4CalcularPalabras()

```
def procesamiento4CalcularPalabras()
    self.palabrasComunesNOODIO=self.delitosnodio[0].split()
    for noodio in self.delitosnodio
        self.palabrasComunesNOODIO=intersección(noodio.lower().split(),self.palabrasComunesNOODIO)
    Fin for
    self.palabrasComunesODIO = self.delitosodio[0].lower().split()
    for odio in self.delitosodio
        self.palabrasComunesODIO=intersección(nodio.lower().split(), self.palabrasComunesODIO)
    Fin for

    self.exclusivasODIO= self.palabrasComunesODIO - self.palabrasComunesNOODIO
    self.exclusivasNOODIO= self.palabrasComunesNOODIO - self.palabrasComunesODIO

    for odio in self.delitosodio
        textoGeneralOdio=textoGeneralOdio+odio
    Fin for
    for noodio in self.delitosnodio
        textoGeneralNOOdio=textoGeneralNOOdio+noodio
    Fin for

    contadorNOOdio=Counter(textoGeneralNOOdio.lower().split())
    contadorOdio=Counter(textoGeneralOdio.lower().split())
    for palabra in contadorOdio.most_common(nComunes)
        if tenerEnCuentaStopWords==False
            if palabra[0] not in stopwords
                self.masRepetidasODIO.append(palabra)
            Fin if
        else
            self.masRepetidasODIO.append(palabra
        Fin if
    Fin for
```

```
    for palabra in contadorNOOdio.most_common(nComunes):
        if tenerEnCuentaStopWords==False
            if palabra[0] not in stopwords:
                self.masRepetidasNOODIO.append(palabra)
            Fin if
        else
            self.masRepetidasNOODIO.append(palabra)
        Fin if
    Fin for

    for elemento in self.masRepetidasODIO
        self.palabrasRepetidasODIONODIO.append(elemento[0])
    Fin for
    for elemento in self.masRepetidasNOODIO
        self.palabrasRepetidasODIONODIO.append(elemento[0])
    Fin for
Fin funcion
```

Ilustración 55: Pseudocódigo de procesamiento4CalcularPalabras()

En esta función se calculan las palabras que aparecen en todos los textos de cada tipo de denuncia, las palabras que son exclusivas de cada tipo de denuncia y las palabras que más se repiten para cada tipo de denuncia. Posteriormente, se almacenan en una bolsa de palabras a la espera de ser utilizada para el borrado.

Primero, para obtener las palabras que aparecen en todos los textos de cada tipo de denuncia, se realiza una intersección iterativa del conjunto de palabras del texto con el conjunto de palabras comunes a los documentos de anteriores iteraciones. Todo ello mediante dos bucles con almacenamiento en sus respectivos vectores, uno para cada tipo de denuncia.

Por otro lado, el cálculo de las palabras exclusivas se realiza mediante una resta de los vectores de palabras comunes de cada denuncia menos las palabras comunes de la denuncia contraria. Es decir, las exclusivas de delitos de Odio son calculadas como *palabrasComunesOdio* - *palabrasComunesNoOdio*, mientras que las exclusivas de delitos de No Odio son calculadas como *palabrasComunesNoOdio* - *palabrasComunesOdio*.

En el último paso, para el cómputo de las palabras más repetidas en cada tipo de denuncia se construyen dos cadenas de texto globales que contienen todas las denuncias de su respectivo tipo. Seguidamente, se realiza un conteo mediante la función Counter. Finalmente, nos quedamos con las *numComunes* palabras más repetidas en cada texto global. Se tendrán en cuenta las stop words en función de la variable *tenerEnCuentaStopWords* mediante una serie de condiciones.

5.5.4.6. Procesamiento4AplicarPalabras()

```
funcion procesamiento4AplicarPalabras(listaTextos)
    listaTextosAuxiliar=listaTextos
    i = 0
    for texto in listaTextos
        for letra in texto
            if letra not in string.punctuation
                lista.append(letra)
            Fin if
        Fin for
        texto = unir(lista)
        palabras = texto.split()
    Fin for
        for palabra in palabras
                    if palabra.lower() not in self.exclusivasNOODIO and palabra not in self.exclusivasODIO
                        if palabra.lower() not in self.palabrasRepetidasODIONODIO
                            if palabra.lower() not in self.palabrasComunesODIO and palabra.lower() not in self.palabrasComunesNOODIO
                                listaAlfa.append(palabra.lower())
        textoAlfa = unir(listaAlfa)

        listaTextosAuxiliar[i]=textoAlfa
        i = i + 1
    return listaTextosAuxiliar
Fin función
```

Ilustración 54: Pseudocódigo de procesamiento4AplicarPalabras()

Esta función se encarga de eliminar de los textos todas las palabras calculadas en la anterior función. En primera instancia, se siguen los mismos pasos que en el *procesamiento2()*. Sin embargo, en el paso final, mediante un bucle con una serie de condiciones, sólo se almacenan en el vector de palabras.

Como consecuencia, nos quedamos con las palabras que no pertenecen a los siguientes conjuntos: Palabras que no son comunes a todos los delitos de Odio, palabras que no son comunes a todos los delitos de No Odio, palabras que no son exclusivas de los delitos de Odio, palabras que no son exclusivas de los delitos de No Odio, palabras que no son las 500 más repetidas en delitos de Odio y las palabras que no son las 500 más repetidas en delitos de Odio.

5.5.4.7. Procesamiento2PorParametro()

```
función procesamiento2PorParámetro(listaTextos)
    listaAuxiliar=listaTextos
    i=0
    for delito in listaTextos
        for letra in delito
            if letra not in string.punctuation
                lista.append(letra)
            Fin if
        Fin for
        texto = unir(lista)
        palabras = texto.cortar()
        listaAlfa = []
        for palabra in palabras
            if not palabra.isupper()
                if palabra.isalpha() or palabra.isnumeric() and not palabra.isupper()
                    listaAlfa.append(palabra)
                Fin if
            Fin if
        Fin for
        textoAlfa = unir(listaAlfa)
        listaAuxiliar[i] = textoAlfa
        i = i + 1
    return listaAuxiliar
Fin función
```

Ilustración 57: Pseudocódigo de procesamiento2PorParametro()

Se trata del *procesamiento2()* adaptado al tratamiento de una sola lista de textos recibida por parámetro, ergo tiene el mismo funcionamiento. Esta función es necesaria para poder recibir parámetros desde el controlador de la web sin estar sujeta a los archivos del corpus.

5.6. Implementación de la web.

Para la construcción de las vistas encontramos varios ficheros *.html*.

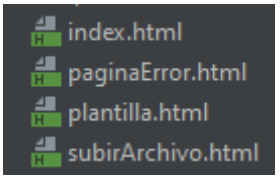

Ilustración 58: Estructura de la web()

El fichero plantilla se trata del fichero base del que beben el index *escribir Texto.html* y *subirArchivo.html*. En este fichero plantilla se han codificado las partes generales de la página como la cabecera y el footer. Para la parte del body se ha generado un bloque de flask:

- En plantilla.html se define *{% block body %} {%endblock %}* vacío para su posterior llenado en las vistas hijas.

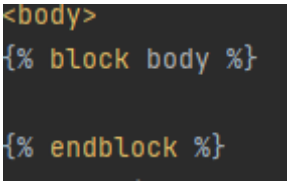

Ilustración 59: Uso de Block en FLask (1)

- En index.html y en *subirArchivo.html* se carga la plantilla mediante un *{% extends index.html %}* y posteriormente en otro bloque se programa todo el contenido específico de cada vista *{% block body %} Contenido específico {% endblock %}*

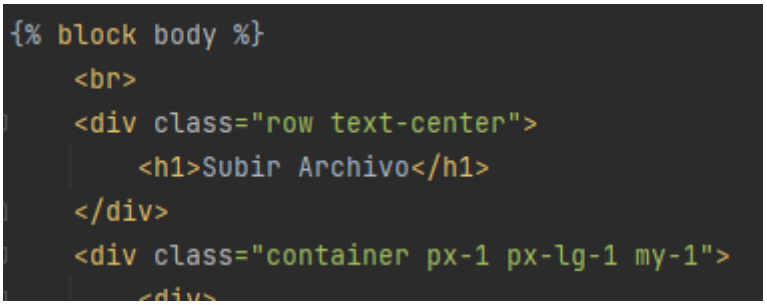

Ilustración 60: Uso de Block en FLask (2)

- Para la gestión de mensajes de error y mensajes de clasificación se ha utilizado los comandos de condición de flask *{% if %} {% else %} {% endif %}*. Donde si el mensaje recibido en la vista en la clase data estaba vació no se mostraba nada.

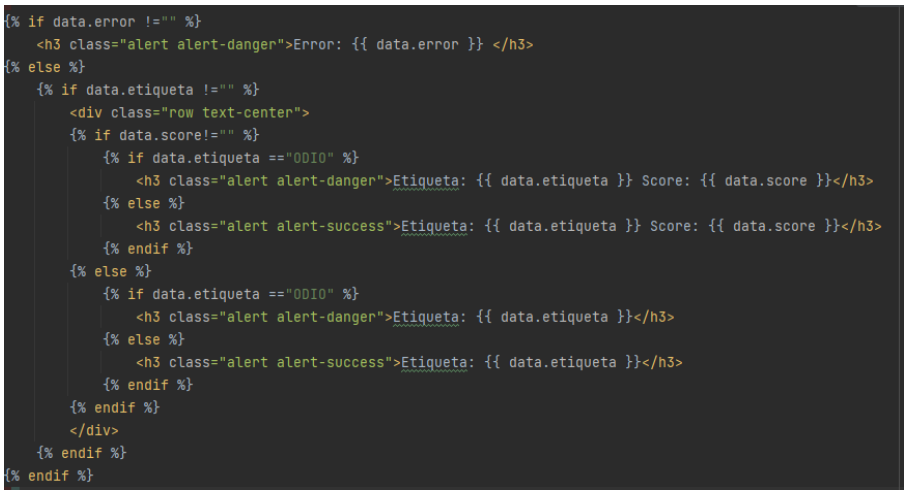

Ilustración 61: Uso de condiciones en Flask

Contaban con mayor prioridad los mensajes de error. En caso de no haber mensaje de error y contar con un mensaje de clasificación (etiqueta y score) no vació, se mostraba en su respectivo color identificativo.

5.6.1. Clase ControladorPagina

5.6.1.1. EscribirTexto()

```
recibirRuta(/, métodos=GET,POST)
recibirRuta(/index.html, métodos=GET,POST)
funcion escribirTexto()
    os.cambiarDirectorio("ModelosDefinitivos")
    listaModelos=os.listarDirectorio
    os.cambiarDirectorio(..)

    if solicictud=POST
        texto = solicitud.form.get('texto')
        modelo = solicitud.form.get('modelo')
        datos['listaModelos']=Clasificador.setUltimoModeloUsado(listaModelos,modelo)
        datos['modelo'] = modelo
        print("modelo",modelo)
        if texto!="":
            etiqueta,score = Clasificador.clasificar(texto,modelo)
            datos['etiqueta']=etiqueta
            datos['score']=score
        fin if
        else:
            datos['error'] = "Debe introducir un texto"
        Fin if
    Fin if
    return renderizar(index.html, datos)
Fin función
```

Ilustración 62: Pseudocódigo de escribirTexto()

Mediante este método el controlador de la página recibe las solicitudes enviadas a la ruta raíz *(/)* tanto como a */index.html.*

En primer lugar se almacena el listado de modelos disponibles obtenido mediante la librería Os en un vector. Seguidamente, si se trata de una petición POST, extraemos el texto y el modelo recibido en la petición.

Si el texto recibido está vacío, enviaremos un mensaje de error. En caso contrario, se invocará al clasificador y se obtendrá la etiqueta junto con el score del texto categorizado. En una instancia de tipo datos iremos almacenando la información que queremos mandar a la vista: modelo, lista de modelos, etiqueta, score…

5.6.1.2. SubirArchivo()

```
recibirRuta(/subirArchivo.html, métodos=GET,POST)
funcion subirArchivo()
    os.cambiarDirectorio("ModelosDefinitivos")
    listaModelos=os.listarDirectorio
    os.cambiarDirectorio(..)

    if solicictud=POST
        archivo = request.files['file']
        modelo = request.form.get('modelo')
        datos['listaModelos'] = Clasificador.setUltimoModeloUsado(listaModelos, modelo)
        envio['modelo'] = modelo
        formatoValido=false
        if archivo.nombre!="":
            archivo.guardarEnDirectorio()
            for archivo in glob.(.doxc)
            open(archivo)
                texto=doc2txt.procesar(archivo)
                if texto != ""
                    etiqueta,score = Clasificador.clasificar(texto,modelo)
                    datos['etiqueta']=etiqueta
                    datos['score']=score
                Fin if
                formatoValido = True
                os.remove(archivo.filename)
            Fin open
            if formatoValido == False:
                datos['error'] = "Solo se acepta formato .docx"
        else:
            datos['error'] = "Debe introducir un archivo .docx"
        Fin if
    Fin if
    return renderizar(subirArchivo.html, datos)
Fin función
```

Ilustración 63: Pseudocódigo de subirArchivo()

Mediante este método el controlador de la página recibe las solicitudes enviadas a la ruta raíz *subirArchivo.html*

En primer lugar, se almacena el listado de modelos disponibles obtenido mediante la librería *Os* en un vector. Seguidamente, si se trata de una petición POST, extraemos el archivo y el modelo recibido en la petición.

Si no se ha recibido ningún archivo se enviará un mensaje de error. En caso contrario, se procederá a abrir el archivo recibido. Si no se puede abrir el archivo. se enviará un mensaje de error de formato incorrecto.

Una vez abierto el archivo, se extrae el texto y se llama al clasificador para obtener la etiqueta con la que ha sido categorizada y su score. En una instancia de tipo *datos* iremos almacenando la información que queremos mandar a la vista: modelo, lista de modelos, etiqueta, score…

5.6.2. Clase Clasificador

5.6.2.1. setUltimoModeloUsado()

```
funcion setUltimoModeloUsado(lista,modelo)
    lista.remove(modelo)
    listaAuxiliar=[]
    listaAuxiliar.append(modelo)
    for elemento in lista:
        listaAuxiliar.append(elemento)
    Fin for
    return listaAuxiliar
Fin función
```

Ilustración 64: Pseudocódigo de setUltimoModeloUsado()

Se borra la instancia en la lista del modelo que estamos usando. Seguidamente, se inserta en la primera posición de una lista auxiliar este modelo. Posteriormente, se copian todos los elementos de la primera lista en la lista auxiliar.

5.6.2.2. Clasificar()

```
función clasificar()
    if modelo[0]=="S" or modelo[0]=="N"
        return clasificarNOBERT(texto, modelo)
    else
        return clasificarBERT(texto, modelo)
    Fin if
Fin función
```

Ilustración 65: Pseudocódigo de clasificar()

En primer lugar, nos situamos en el directorio de Modelos definitivos e instanciamos un pipeline de transformers con nuestro modelo entrenado.

Una vez recibido el resultado de la clasificación en una cadena, extraemos ciertas posiciones de esa cadena. En función de la tipología de denuncia obtenida, formateamos como ODIO o NO ODIO. Para el score simplemente nos quedamos el fragmento de cadena en el que se encuentra el score.

5.6.2.3. ClasificarBERT()

```
funcion clasificarBERT(texto, modelo)
    os.cambiarDirectorio("ModelosDefinitivos")
    clasificador=transformers.pipeline("text-classification", truncation=True,
                                       tokenizer=AutoTokenizer.from_pretrained(modelo+"\\"+checkpoint), batch_size=16,
                                       model=modelo+"\\"+checkpoint)
    os.cambiarDirectorio("..")

    listaAuxiliar.append(texto)
    procesador = procesaTexto(listaAuxiliar, listaAuxiliar)
    textoProcesado=procesador.procesamiento2PorParámetro(listaAuxiliar)[0]

    salida = string(classifier(textoProcesado)).split()

    if salida[1] == "'NO"
        etiqueta = "NO ODIO"
        string(salida[4]).replace("}", "")
        string(salida[4]).replace("]", "")
        score = string(salida[4]).replace("}", "")
        score = score.replace("]", "")
    else
        etiqueta = "ODIO"
        score = str(salida[3]).replace("}", "")
        score = score.replace("]", "")
    Fin if
    return etiqueta, score
Fin función
```

Ilustración 66: Pseudocódigo de clasificarBERT()

Primero, nos situamos en el directorio de *ModelosDefinitivos* y creamos un pipeline transformers con nuestro modelo entrenado.

Posteriormente, llamamos a la función de *procesamiento2()* por parámetro, tratamos el texto a clasificar e invocamos el método clasificador.

Una vez recibido el resultado de la clasificación en una cadena, extraemos ciertas posiciones de esa cadena y en función de la tipología de denuncia obtenida formateamos como ODIO o NO ODIO. Para el score simplemente nos quedamos el fragmento de cadena en el que se encuentra el score

5.6.2.4. ClasificarNoBERT()

```
función clasificarNoBERT(texto,modelo)
    os.cambiarDirectorio("ModelosDefinitivos")
    os.chdir(modelo)

    contenidoCarpeta=os.listdir(os.getcwd())
    model = Joblib.load(contenidoCarpeta[0])
    vectorizer=Joblib.load(contenidoCarpeta[1])

    listaAuxiliar.append(texto)
    procesador = procesaTexto(listaAuxiliar, listaAuxiliar)
    textoProcesado = procesador.procesamiento2PorParámetro(listaAuxiliar)[0]

    def tokenizar(texto)
        texto = Spacy.nlp(texto)
        for token in texto:
            if token.text.isalpha() and not token.is_stop
                tokens.append(token.lemma_.lower())
        textoProcesado = unir(tokens)
        return textoProcesado

    textoProcesado=tokenizar(textoProcesado)

    prediction = model.predict(vectorizer.transform([textoProcesado]))
    if prediction==1
        etiqueta = "NO ODIO"
        score = ""
    else
        etiqueta = "ODIO"
        score = ""
    Fin if
    return etiqueta, score
Fin función
```

Ilustración 67: Pseudocódigo de clasificarNoBERT()

Como primer paso, nos movemos al directorio de Modelos definitivos y nos situamos en el directorio del modelo recibido por parámetro. Seguidamente, listamos el contenido de este directorio. El primer elemento corresponderá con el modelo y el segundo con el vectorizador.

A continuación, llamamos a la función de procesamiento 2 con el texto a tratar y lo sometemos a la misma tokenización explicada en *entrenarBERT()*.

Finalmente, se realiza una predicción del texto y en función del número recibido pertenece a una etiqueta u otra. 1 para delitos de No Odio y 0 para delitos de Odio.

5.7. Resultados Obtenidos

Tal y como se ha indicado en el diseño de la página web tanto la cabecera como el footer son comunes a ambas vistas. Tal y como se ha explicado anteriormente, esto ha sido implementado mediante la definición de una plantilla.html con estos componentes haciendo uso de las funciones block de *Flask*.

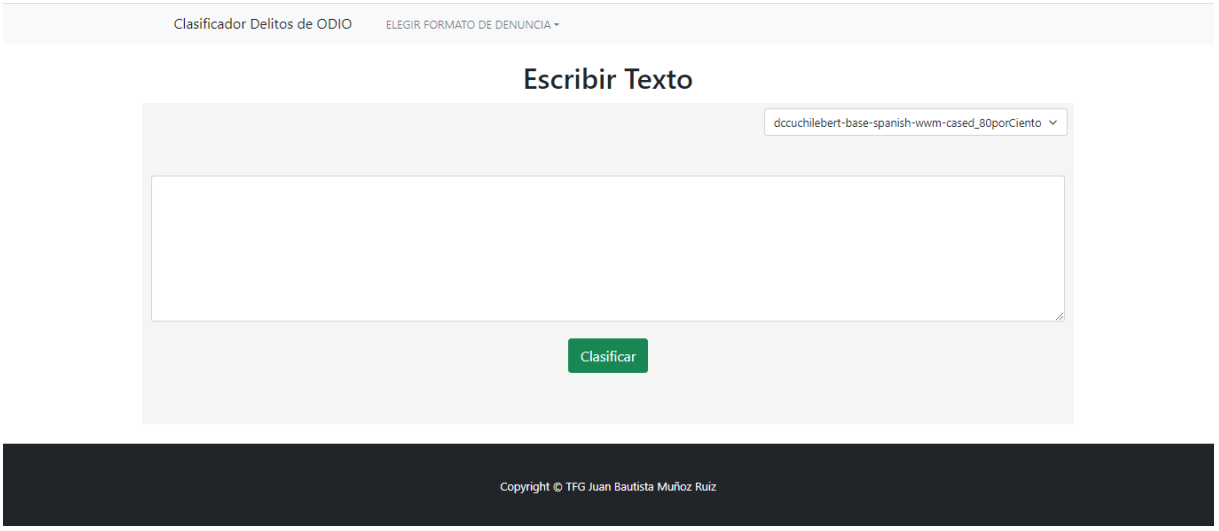

Ilustración 68: Vista Escribir Texto

Así quedaría la página para escribir texto. Donde en el cuerpo de la página podemos escribir un texto, seleccionar el modelo a usar en la predicción y pulsar el botón clasificar.

Ilustración 69: Vista Subir Archivo

En cuanto a la página subir archivo, el funcionamiento es el mismo solo que cambia el campo de texto por un formulario de subida de archivo.

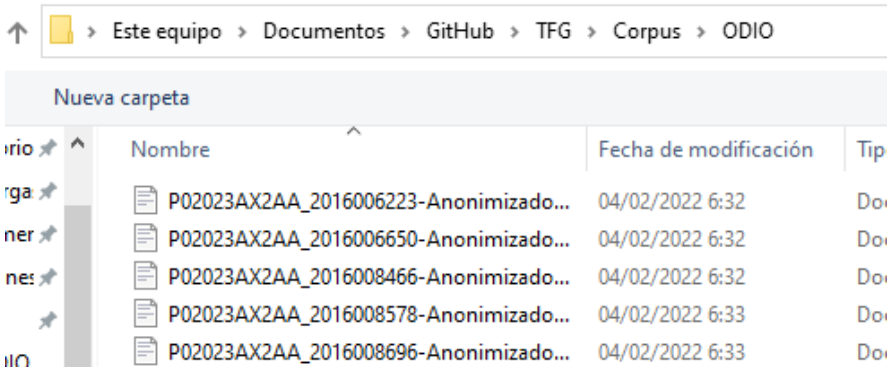

Ilustración 70: Subida de archivo

Al clicar el subir archivo se despliega el menú de carpetas de nuestro sistema operativo en el que podremos elegir el archivo a subir.

Ilustración 71: Selección de formato de denuncia

Por otro lado el usuario podrá elegir qué tipo de formato de archivo quiere clasificar, si formato textual o formato archivo.

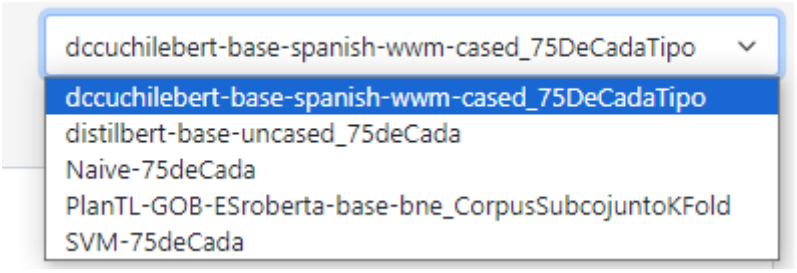

Ilustración 72: Selección de modelo

El botón de selección de modelo queda tal que así. Se nos muestran los diferentes modelos presentes en nuestra carpeta *ModelosDefinitivos* de entre los que podremos elegir cúal vamos a usar.

A continuación, la visualización de los diferentes mensajes en la página según la información recibida desde el controlador:

- Mensaje de denuncia clasificada como NO ODIO:

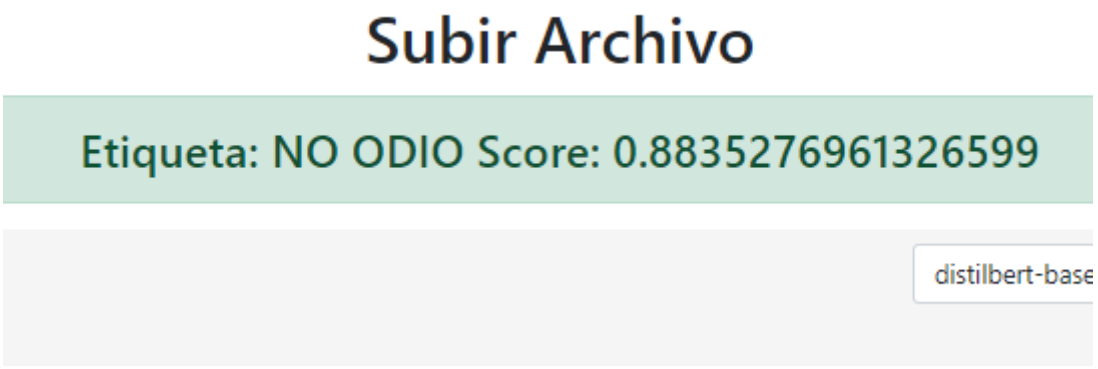

Ilustración 73: Denuncia clasificada como NO ODIO

- Mensaje de denuncia clasificada como ODIO:

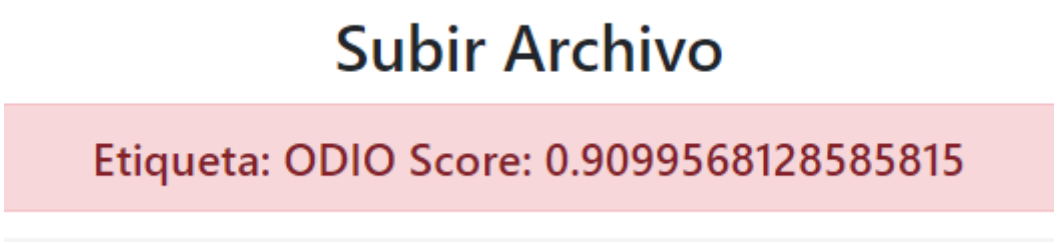

Ilustración 74: Denuncia clasificada como ODIO

- Mensaje de fallo en subida de archivo:

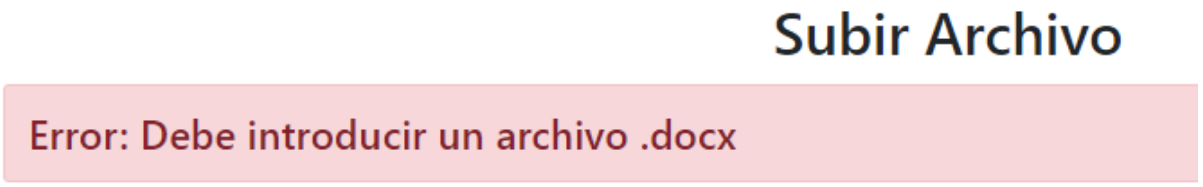

Ilustración 75: Mensaje de error, archivo no introducido

- Mensaje de fallo al subir un archivo en un formato diferente al *.docx*:

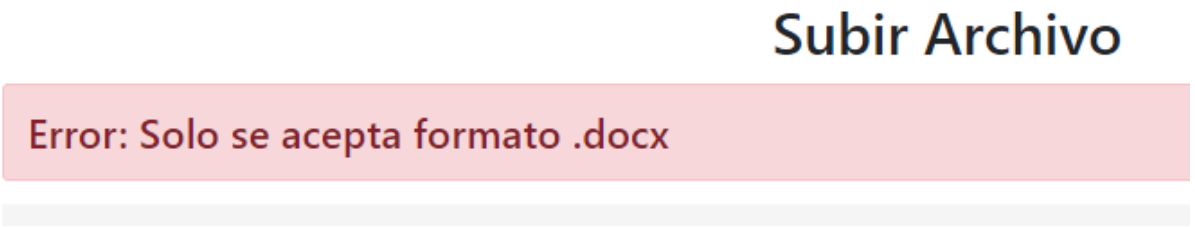

Ilustración 76: Mensaje de error, formato de archivo incorrecto

6. PRUEBAS Y RESPUESTA DEL SISTEMA

En esta sección se explican las pruebas de caja blanca y las de caja negra realizadas sobre el sistema.

6.1. Pruebas de caja blanca

Una vez implementado el sistema se han realizado una serie de pruebas para detectar errores en el código.

En primer lugar, se ha comprobado el recorrido de todos los bucles. Una prueba de caja blanca en la que se ha validado que todas las condiciones son accedidas para cada casuística en función de las diferentes variables booleanas o numéricas establecidas como condición. Además, en cada momento se visualizaba en pantalla en qué rama de cada bucle estaba iterando el sistema.

Seguidamente, se realizó un chequeo de las diferentes vistas de la web mediante un método de análisis basado en la correcta visualización desde el

navegador junto con el funcionamiento del código y la comunicación entre la vista web y el controlador.

Por otro lado, encontramos la validación de la correcta asignación de las variables y construcción de estructuras de datos. Todo ello mediante una serie de test durante el desarrollo del software en las cuales se iba mostrando por consola el contenido de las diferentes variables y conjuntos de variables.

Otras de las principales pruebas realizadas sobre el sistema, fueron las de evaluación de los diferentes modelos clasificadores. Todo ello de cara a elegir los mejores para ser incluidos en la página web.

Para finalizar, se realizaron unas pruebas manuales para comprobar la correspondencia entre los resultados de la validación cruzada y los obtenidos en la clasificación mediante la web. Pruebas de vital importancia para comprobar el correcto funcionamiento tanto de los modelos clasificadores como la propia veracidad de los resultados obtenidos.

6.2. Pruebas de caja negra

Una vez implementada la interfaz web, se trabajó en la verificación del correcto funcionamiento de la visualización de mensajes y navegabilidad.

En primer lugar, se comprobaron los mensajes de error mediante una serie de ejecuciones. En estas ejecuciones se iba observando la correcta generación de los mensajes de error para los casos de subida de un archivo en formato distinto del *.docx* en la vista *subirArchivo* y para los casos en los que en la vista *escribirTexto* no se insertaba ningún carácter.

A continuación, se visualiza el proceso seguido en la prueba de subida de archivo erróneo con un fichero *.pdf*:

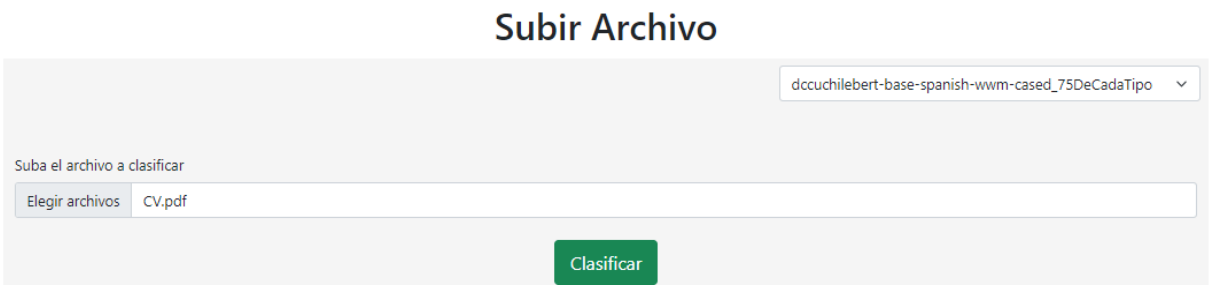

Ilustración 77: Prueba de subida archivo erróneo, parte 1.

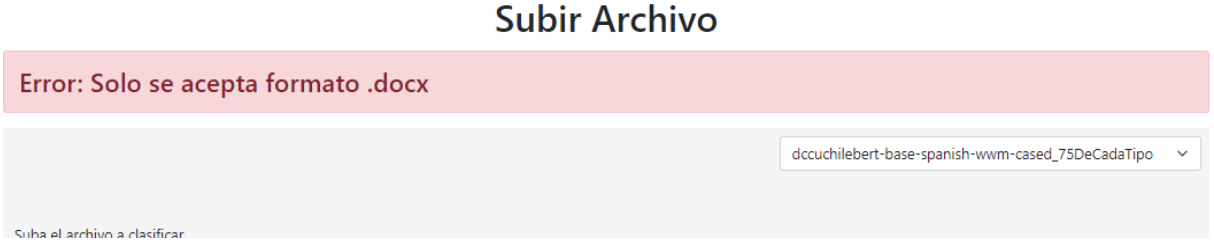

Ilustración 78: Prueba de subida archivo erróneo, parte 2.

Seguidamente, se visualiza el proceso seguido en la prueba de escritura vacía:

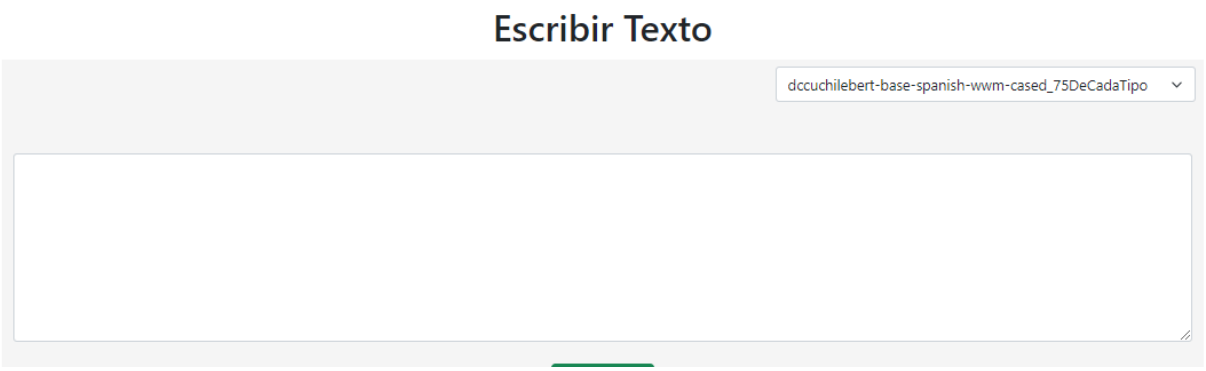

Ilustración 79: Prueba de escritura vacía, parte 1.

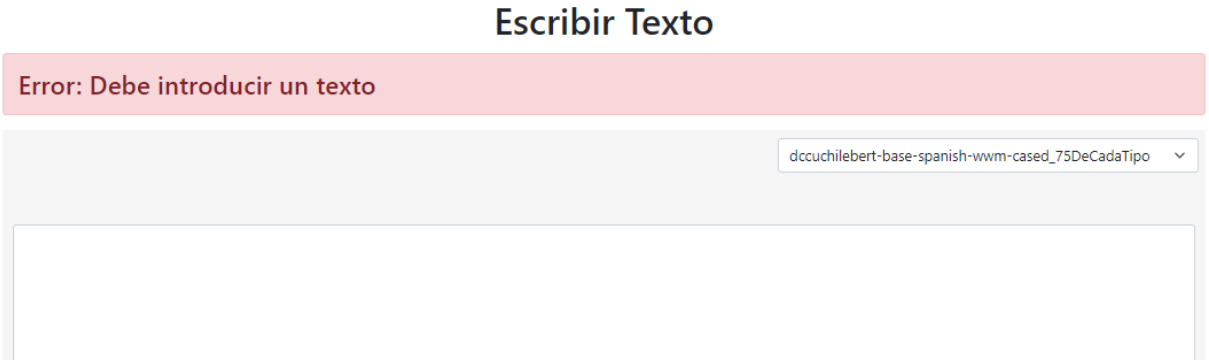

Ilustración 80: Prueba de escritura vacía, parte 2.

Por otro lado, se aseguró el correcto funcionamiento de la visualización de los mensajes de clasificación según el tipo de denuncia con su respectivo color representativo.

En este ejemplo gráfico, se muestra el proceso de prueba de una denuncia de ODIO:

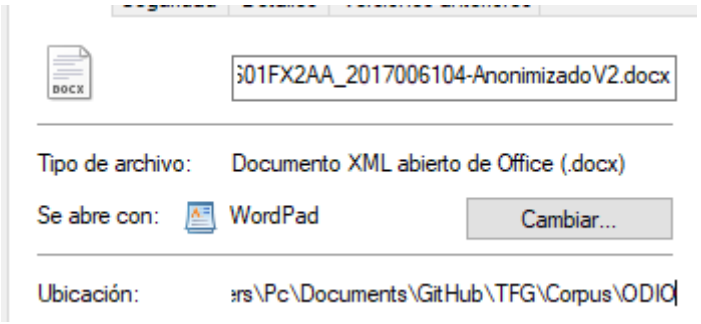

Ilustración 81: Prueba de visualización de mensajes, parte 1.

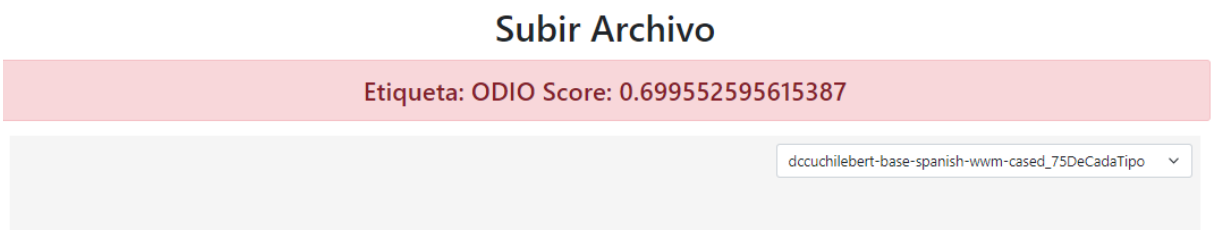

Ilustración 82: Prueba de visualización de mensajes, parte 2.

Asimismo, se realizó un test de navegabilidad sobre la web en el que se comprobó que todas las vistas estuvieran conectadas y accesibles entre sí.

Finalmente, se compró el despliegue de la pestaña de modelos asegurando la visualización de todos los modelos disponibles. En esta prueba se comprobaba que todos los modelos ubicados en la carpeta */TFG/ModelosDefinitivos* estuvieran presentes en la pestaña de selección de modelos de la web.

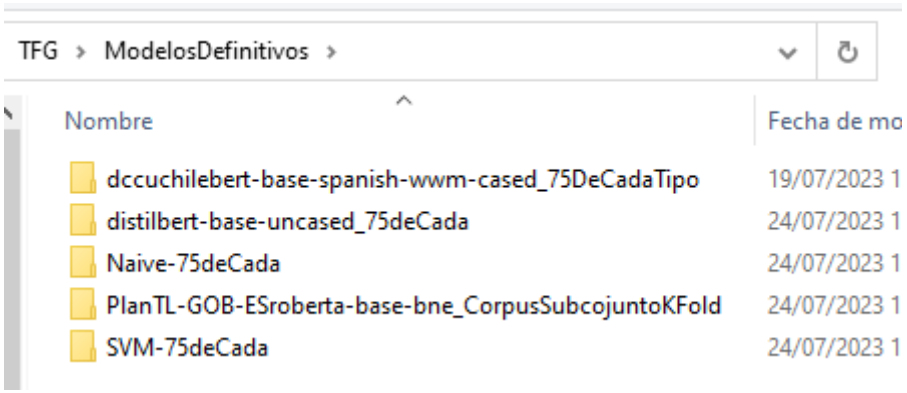

Ilustración 83: Prueba de la pestaña modelos, parte 1.

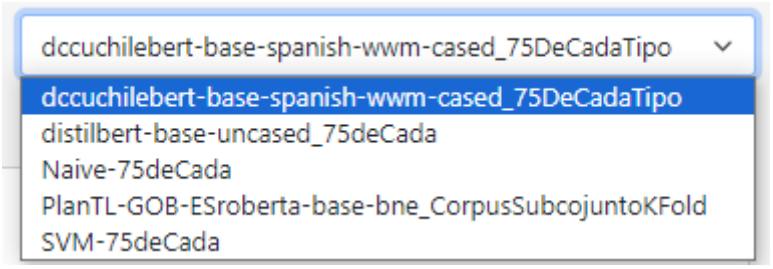

Ilustración 84: Prueba de la pestaña modelos, parte 2.

7. ANÁLISIS DEL SISTEMA DE CLASIFICACIÓN

En este apartado procederemos a analizar los resultados del sistema clasificador. Todo ello con el objetivo de entender los resultados de alto rendimiento generados y dotar de validez a estos.

El hecho de contar con una estructura bastante definida y diferenciada entre cada tipo de denuncia hace que la gran mayoría de modelos obtengan resultados

bastante altos. En la siguiente ilustración, se puede vislumbrar claramente la frontera entre ambos tipos de denuncia.

```
ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P06085AX2AA_2017016598.pdf AT< Instructor:
ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P07391LX2AA_2017009364.pdf AT< Instructor:
ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P07601EX2AA_2016012236.pdf AT< Instructor:
ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P07601FX2AA_2016014670.pdf AT< Instructor:
ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P07601FX2AA_2016018028.pdf AT< Instructor:
ANONIMIZADO_0500_Caso_C0740398_obj_78695538.PDF Atestado: AT<<<<< Instructor: NUM<<< Secretario:
ANONIMIZADO_0500_Caso_C0740401_obj_79053671.PDF Atestado: AT<<<<< Instructor: NUM<< Secretario: X
ANONIMIZADO_0500_Caso_C0740403_obj_78649456.PDF Atestado: AT<<<<< Instructor: NUM<<< Secretario:
ANONIMIZADO_0500_Caso_C0740419_obj_78680912.PDF Atestado: AT<<<<< Instructor: NUM<< Secretario: A
ANONIMIZADO_0500_Caso_C0740429_obj_78683075.PDF Atestado: AT<<<<<< Instructor: NUM<<< Secretario:
```

Ilustración 85: Estructura de las denuncias (1)

Como podemos observar, cada tipo de documento está bastante definido en un formato concreto. Las cinco primeras líneas se tratan de denuncias de Odio y las cinco últimas de denuncias de No Odio.

Incluso si aplicamos una capa extra de procesamiento eliminando números y carácteres no numéricos, vislumbramos de una forma más clara las diferencias entre cada tipo de denuncia.

```
Val Instructor Secretario Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de qui
Val Instructor Secretario Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de qui
Val Instructor Secretario Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de qui
Val Instructor Secretario Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de qui
Val Atestado Instructor Secretario Atestado n° Dependencia Se extiende a las 02 horas 03 minutos del día 2019 para hacer con
Val Atestado Instructor Secretario Atestado n° Dependencia En Huelva siendo las 01 horas 07 minutos del día 2019 ante el Ins
Val Atestado Instructor Secretario Atestado n° Dependencia Se extiende en siendo las 02 horas 39 minutos del día 2019 por el
Val Atestado Instructor Secretario Atestado n° Dependencia En Sevilla siendo las 02 horas 57 minutos del día 2019 ante el In
```

Ilustración 86: Estructura de las denuncias (2)

Como consecuencia, las métricas de los modelos son bastante buenas, ya que los modelos tienen gran facilidad para diferenciar entre un tipo de denuncia u otra.

A continuación, una comparativa del rendimiento de los diferentes modelos probados. Todas las mediciones realizadas se encuentran en el archivo *Media_métricas* dentro la carpeta *TFG/Métricas* ubicada en el *.zip* entregado. Por cada modelo encontramos un archivo csv con sus resultados propios. Para el análisis de la validación cruzada nos centraremos en archivo Métricas/media_Metricas.csv.

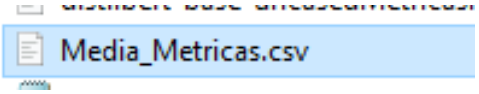

Ilustración 87: Archivo de métricas de la validación cruzada

Los siguientes análisis de diferentes tratados y procesamientos de textos fueron realizados con una validación k-fold con *k=3* utilizando todo el corpus de denuncias. Es decir, 110 atestados de Odio y 80 de No Odio.

Cabe destacar que las validaciones no fueron realizadas con una única semilla con el objetivo de desacoplar los resultados de la distribución de las carpetas. Es decir, para invitar que divisiones del corpus en entrenamiento y validación propensas a obtener peores resultados no influyeran negativamente en los resultados.

7.1. Métricas con procesamiento 1 (k=3)

Descripción del procesamiento:

- Elimina espacios vacíos irrelevantes del formateo .docx.

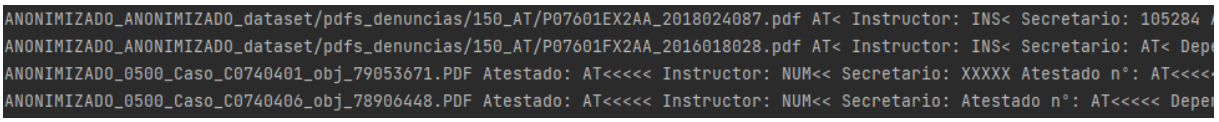

Ilustración 88: Visualización del procesamiento 1

Modelo	Nombre clave en fichero de parámetros	macro-precision	macro-recall	macro-f1	Tiempo (Segundos)
SVM	SVM	1	1	1	462.49424934
Naive Bayes	Naive	1	1	1	461.920893666
Distilbert	distilbert-base-uncased	1	1	1	2091.344172477722
Beto	dccuchile/bert-base-spanish-wwm-cased	1	1	1	6711.93279838562
Maria	PlanTL-GOB-ES/roberta-base-bne	1	1	1	4865.794182777405
Bertin	bertin-project/bertin-roberta-base-spanish	1	1	1	5012.07196521759
Multilingual Bert	bert-base-multilingual-cased	1	1	1	6701.025990009308

Tabla 26: Comparación de métricas con procesamiento 1

Se consiguen métricas de muy alto rendimiento con valores al máximo. Para tratar de producir resultados más realistas en el siguiente procesamiento se eliminará la primera línea de la denuncia. Esta línea indica el nombre del documento y para cada tipo de denuncia sigue un mismo patrón bien diferenciado con respecto a la clase contrario. Como consecuencia, la mera presencia de esta frase podría estar adelantando de a qué tipo de clase pertenece haciendo la labor de etiqueta indirecta. Esta característica la podemos observar en las siguientes ilustraciones:

ANONIMIZADO_ANONIMIZADO_dataset/pdfs_denuncias/150_AT/P02023AX2AA_2016006223.pdf

Ilustración 89: Primera línea en denuncias de Odio

ANONIMIZADO_0500_Caso_C0740394_obj_78677408.PDF

Ilustración 90: Primera línea en denuncias de No Odio

7.2. Métricas con procesamiento 2 (k=3)

Descripción del procesamiento:

- Elimina espacios vacíos irrelevantes del formateo .docx.
- No se tienen en cuenta los carácteres no alfabéticos:
  - Cabecera con el nombre del documento.
  - Etiquetas de anonimizado.
  - Elimina los signos de puntuación.

```
Instructor Secretario 105284 Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de quien
Instructor Secretario Dependencia En siendo las del ante el Instructor y Secretario arriba mencionados En calidad de quien median
Atestado Instructor Secretario Atestado n° Dependencia En calidad de denunciante quien mediante n° y número de soporte acredita s
Atestado Instructor Secretario Atestado n° Dependencia En Huelva siendo las 03 horas 00 minutos del día 2019 ante el Instructor y
```

Ilustración 91: Visualización del procesamiento 2

Modelo	Nombre clave en fichero de parámetros	macro-precision	macro-recall	macro-f1	Tiempo (Segundos)
SVM	SVM	0.9933333333333333	0.9966666666666667	0.9933333333333333	164.72374320030212
Naive Bayes	Naive	0.94	0.9533333333333333	0.9433333333333334	163.61337757110596
Distilbert	distilbert-base-uncased	0.9266666667	0.8533333333	0.8533333333	2169.367501
Beto	dccuchile/bert-base-spanish-wwm-cased	0.9933333333	0.9966666667	0.9933333333	7165.572788

Maria	PlanTL-GOB-ES/roberta-base-bne	0.946667	0.913333	0.920000	5077.588089466095
Bertin	bertin-project/bertin-roberta-base-spanish	0.836667	0.726667	0.726667	5492.295664
Multilingual Bert	bert-base-multilingual-cased	0.4666666667	0.5333333333	0.4266666667	6841.58885

Tabla 20: Comparación de métricas con procesamiento 2

Con este procesamiento obtenemos resultados de alta resolución pero con la principal diferencia de que al eliminar la primera frase son más realistas.

7.3. Métricas con procesamiento 3 (k=3)

Descripción del procesamiento:

- Elimina espacios vacíos irrelevantes del formateo .docx.
- No se tienen en cuenta los carácteres no alfabéticos:
  - Cabecera con el nombre del documento.
  - Etiquetas de anonimizado.
- Elimina los signos de puntuación.
- Elimina frase común al tipo de denuncia:
  - ODIO: No tiene en cuenta las palabras desde “Instructor” hasta “ocurridos”
  - NO ODIO: No tiene en cuenta las palabras desde “Atestado” hasta “Dependencia”

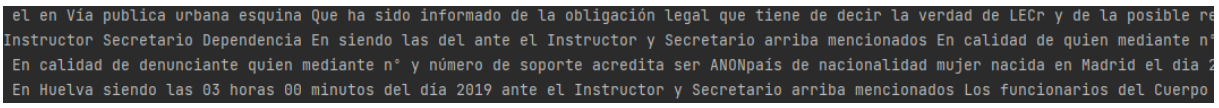

Ilustración 92: Visualización del procesamiento 3

Modelo	Nombre clave en fichero de parámetros	macro-precision	macro-recall	macro-f1	Tiempo (Segundos)
SVM	SVM	0.9833333333333334	0.9833333333333334	0.9833333333333334	110.71968221664429
Naive Bayes	Naive	0.9700000000000001	0.9766666666666666	0.9733333333333333	104.49898099899292
Distilbert	distilbert-base-uncased	0.9366666666666666	0.9	0.9066666666666666	2244.545994758606
Beto	dccuchile/bert-base-sp	0.9766666666666667	0.9633333333333334	0.9666666666666667	7221.718133449554

	anish-wwm-cased				
Maria	PlanTL-GOB-ES/roberta-base-bne	0.9533333333333	0.9366666666666	0.9366666666666	5032.490589857101
Bertin	bertin-project/bertin-roberta-base-spanish	0.9533333333335	0.9566666666667	0.9500000000001	5013.845260620117
Multilingual Bert	bert-base-multilingual-cased	0.8333333333334	0.8266666666667	0.82	7289.187020778656

Tabla 21: Comparación de métricas con procesamiento 3

Este procesamiento nació como un intento de medir hasta qué punto los resultados estaban condicionados por la repetición de una frase común y diferenciada en ambos tipos de denuncias. Sin embargo, se obtienen resultados muy parecidos al procesamiento 2. Se podría concluir que este tratamiento de los textos necesita ser ampliado y mejorado para evaluar en mejor condición esta posible casuística. Esta idea será sometida a análisis en el procesamiento 4.

7.4. Métricas con procesamiento 4 (k=3)

Descripción del procesamiento:

- Elimina espacios vacíos irrelevantes del formateo .docx.
- No se tienen en cuenta los carácteres no alfabéticos:
  - Cabecera con el nombre del documento.
  - Etiquetas de anonimizado.
- Elimina los signos de puntuación.
- Elimina las siguientes palabras de cada tipo de denuncia (Sin tener en cuenta stop words):
  - Palabras comunes de cada clase, es decir, palabras presentes en todos los documentos de una misma clase.
  - Palabras más repetidas de cada clase.
  - Palabras exclusivas de cada clase, es decir, palabras que aparecen en todos los documentos de una misma clase y no aparecen en el otro tipo de denuncia.

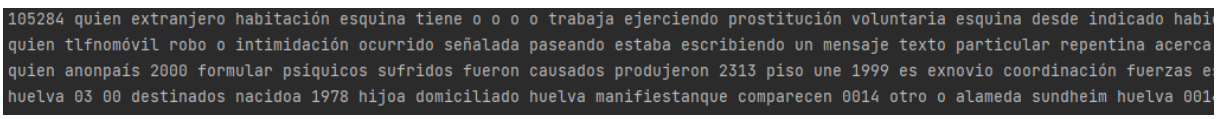

Ilustración 93: Visualización del cesamiento 4a

Modelo	Nombre clave en fichero de parámetros	macro-precision	macro-recall	macro-f1	Tiempo (Segundos)
SVM	SVM	0.96	0.9499999999999998	0.9566666666666667	349.559139251709
Naive Bayes	Naive	0.89	0.8966666666666666	0.8766666666666666	362.44042778015137
Distilbert	distilbert-base-uncased	0.9333333333333332	0.9433333333333334	0.9366666666666665	2559.1892251968384
Beto	dccuchile/bert-base-spanish-wwm-cased	0.8366666666666666	0.8433333333333333	0.84	7209.92800450325
Maria	PlanTL-GOB-ES/roberta-base-bne	0.8566666666666666	0.8633333333333333	0.8533333333333334	5129.316880702972
Bertin	bertin-project/bertin-roberta-base-spanish	0.8966666666666666	0.9033333333333333	0.8966666666666666	5126.048735618591
Multilingual Bert	bert-base-multilingual-cased	0.8933333333333332	0.8433333333333333	0.8333333333333334	7290.594949007034

Tabla 29: Comparación de métricas con procesamiento 4a

Encontramos un descenso de los valores muy poco significativo. Pese a ello, siguen siendo métricas bastante altas en torno al 0,9. Este procesamiento es de gran utilidad para confirmar que las denuncias de cada tipo son de estructuras muy diferenciadas, por lo que el sistema implementado obtendrá altos valores con cualquiera de los procesamientos.

7.5. Métricas con procesamiento 4 con stopwords (k=3)

Descripción del procesamiento:

- Se trata del mismo funcionamiento que el procesamiento 4 pero teniendo en cuenta stop words en la eliminación de palabras comunes repetidas o exclusivas.

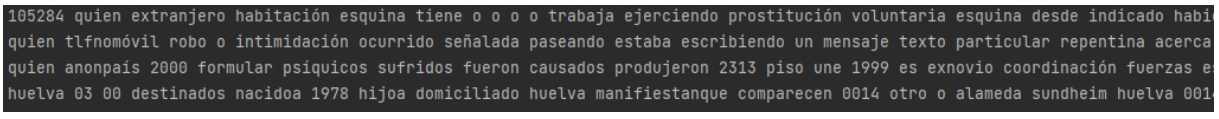

Ilustración 94: Visualización del procesamiento 4b

Modelo	Nombre clave en fichero de parámetros	macro-precision	macro-recall	macro-f1	Tiempo (Segundos)
SVM	SVM	0.9433333333333334	0.9233333333333333	0.9266666666666667	418.4788725376129
Naive Bayes	Naive	0.8966666666666666	0.8999999999999999	0.8833333333333333	353.0910048484802
Distilbert	distilbert-base-uncased	0.9066666666666667	0.9166666666666666	0.9033333333333333	2570.555022954941
Beto	dccuchile/bert-base-spanish-wwm-cased	0.8266666666666667	0.8266666666666667	0.8266666666666667	7188.808485984802
Maria	PlanTL-GOB-ES/roberta-base-bne	0.8533333333333334	0.8666666666666667	0.8566666666666666	5100.20978474617
Bertin	bertin-project/bertin-roberta-base-spanish	0.8466666666666667	0.8466666666666667	0.8366666666666666	5061.293961763382
Multilingual Bert	bert-base-multilingual-cased	0.86	0.87	0.8566666666666666	7369.047768831253

Tabla 30: Comparación de métricas con procesamiento 4b

Se consiguen resultados prácticamente similares o con un descenso de valor de 0,4. Con este procesamiento se intentaba comprobar hasta qué punto las stop words condicionan a generar una estructura diferenciada entre ambos tipos de denuncia.

Podemos concluir que por muchas palabras que se eliminen la tipología de cada denuncia está bien definida y diferenciada en cuanto a formato, haciendo así que el sistema clasificador esté muy cerca de la completitud en predicción de etiquetas correctas.

De cara a obtener el mejor rendimiento en el sistema clasificador nos quedaremos con el procesamiento 2, ya que es el que obtiene resultados más reales y óptimos. Elegiremos este procesamiento para el tratado de los textos tanto en el entrenamiento de los modelos como en su correspondiente uso mediante la web. Los modelos que mejor rendimiento alcanzan son María, Beto y Distilbert.

8. MANUAL DE INSTALACIÓN Y USO

Nos situamos en la carpeta principal entregada, la cual contiene los siguientes elementos.

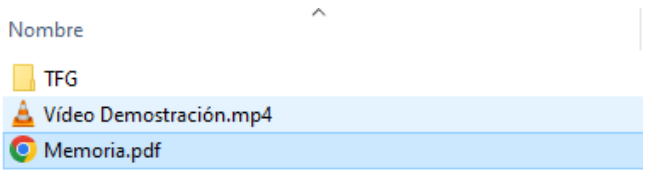

Ilustración 95: Carpeta principal

- TFG: Carpeta con todo el código fuente del proyecto y varios elementos que serán detallados posteriormente.
- Vídeo Demostración: Vídeo en formato mp4 con una demostración del funcionamiento y uso.
- Memoria: Documento en formato pdf con la memoria del proyecto.

Dentro de la carpeta TFG disponemos de los siguientes ficheros:

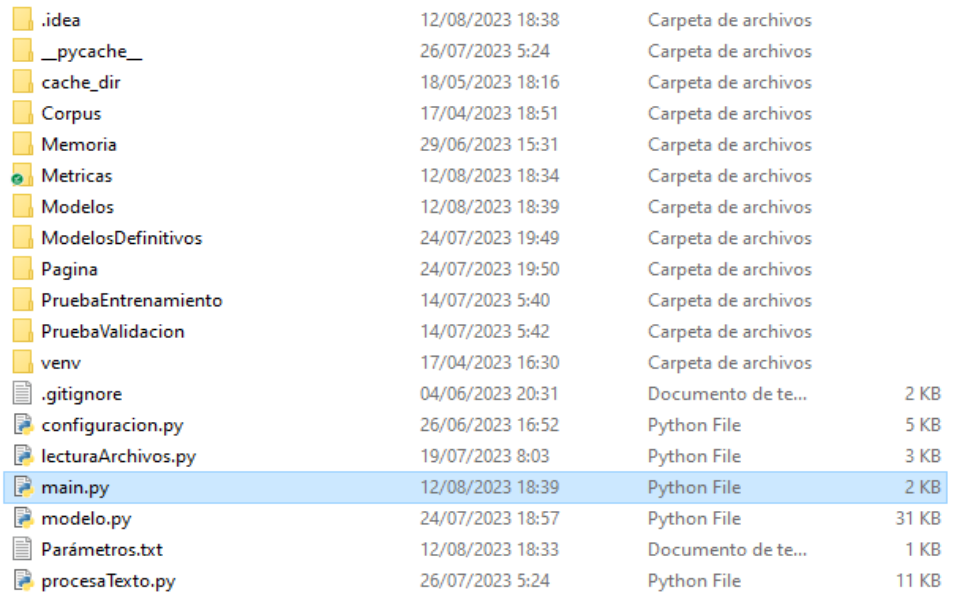

Ilustración 96: Carpeta TFG

Por un lado, encontramos carpetas presentes en el desarrollo y prueba del proyecto que no han podido ser subidas a la plataforma de entrega al no disponer de permiso para ello, como lo son todas las referidas al corpus:

- **Corpus:** Se trata de la carpeta con el conjunto de todas las denuncias policiales. Está dividida en dos carpetas indicando la tipología de los atestados, ODIO y NO ODIO.

- **PruebaEntrenamiento:** Carpeta que contiene el corpus con el que ha sido entrenado el modelo de prueba de resultados de la validación cruzada *PlanTL-GOB-ESroberta-base-bne_CorpusSubcojuntoKFold*.
- **PruebaValidación:** Carpeta que contiene un subconjunto del corpus obtenido en una validación cruzada k-fold para la validación del modelo *PlanTL-GOB-ESroberta-base-bne_CorpusSubcojuntoKFold.*

Por otro lado, encontramos los siguientes archivos y carpetas a destacar. Todas las que no sean nombradas, son propias de Pycharm, de la implementación o carecen de importancia para la instalación y uso:

- **Métricas:** Esta carpeta contiene todos los archivos en formato *.csv* con las métricas de las últimas ejecuciones de validaciones cruzadas sobre cada tipo de modelo. Dentro de esta carpeta la que tiene mayor importancia y en la que nos deberemos fijar es *Media_metricas.csv.*
- **Modelos:** Se trata de la carpeta auxiliar donde se guardan los modelos después de un entrenamiento.
- **ModelosDefinitivos:** Carpeta donde almacenar los modelos finales que serán accedidos desde la web.
- **Página:** Contiene el código fuente de la aplicación web. Dentro de esta el archivo principal para la ejecución es *controladorPágina.py.*
- **Parámetros.txt:** Archivo con las variables de configuración explicadas en el punto 4.3.1.2.
- **main.py:** Fichero de ejecución del sistema clasificador.

Para el uso de la aplicación implementada bastará con abrir el proyecto fuente y ejecutarlo en cualquier entorno Python. Es recomendable que sea PyCharm, en este caso con Python 3.9. Para ejecutar las dos partes de la aplicación habrá que seguir las siguientes instrucciones:

- Lanzar el sistema: Ejecutar el fichero *main.py.*
- Lanzar la web: Ejecutar el fichero que se encuentra en la carpeta Página *controladorPágina.py.*

Para importar los modelos entrenados habrá que seguir los siguientes pasos:

- Si la carpeta ModelosDefinitivos está llena:
  - Los modelos ya están importados. No realizar ninguna acción.
- Si la carpeta ModelosDefinitivos está vacía:
  - Significa que la plataforma de entrega no soporta el peso de los modelos.
  - Realizar la descarga desde el siguiente enlace: Enlace descarga.
  - Extraer el fichero *.zip.*
  - Sustituir la carpeta obtenida *ModelosDefinitivos* por la que estaba vacía.

9. MANUAL DE USUARIO

A continuación, se presentan los manuales de usuario con la información sobre los comandos del fichero de parámetros, pasos a seguir para la ejecución y una descripción de los modelos entrenados disponibles.

9.1. Manual de usuario del clasificador

Parámetro principal:

- Para realizar una validación cruzada deberemos ejecutar el fichero main.py con la siguiente configuración en el fichero Parámetros.txt:
  - **modo=**Validacion
- Para realizar un entrenamiento deberemos ejecutar el fichero main.py con la siguiente configuración en el fichero Parámetros.txt:
  - **modo=**Entrenamiento

Parámetros extra:

- **carpetaCorpus:** Carpeta en la que tenemos almacenado el corpus.
  - Si queremos el corpus completo (80 denuncias No Odio y denuncias de 110 Odio):
    - **carpetaCorpus=**.\Corpus
  - Si queremos el corpus de la última división en la validación k-fold con semilla 42 (55 denuncias de No Odio y 72 denuncias de no Odio).
    - **carpetaCorpus=**.\PruebaEntrenamiento
- **modeloPreentrenado:** Los parámetros a introducir son los siguientes:
  - Para un modelo SVM:
    - **modeloPreentrenado=**SVM
  - Para un modelo Naive Bayes:
    - **modeloPreentrenado=**Naive
  - Para un modelo Distilbert:
    - **modeloPreentrenado=**distilbert-base-uncased
  - Para un modelo Beto:
    - **modeloPreentrenado=**dccuchile/bert-base-spanish-wwm-cased
  - Para un modelo María:
    - **modeloPreentrenado=**PlanTL-GOB-ES/roberta-base-bne
  - Para un modelo Bertín:
    - **modeloPreentrenado=**bertin-project/bertin-roberta-base-spanish
  - Para un modelo Multilingual Bert:
    - **modeloPreentrenado=**bert-base-multilingual-cased
- **comentario:** Cualquier palabra o frase que sirva para diferenciar esta ejecución sobre las demás. Por ejemplo:
  - **comentario=**CualquierFraseIdentificativa

- **k:** Parámetro *k* para la validación k-fold. Debe ser entero. Un ejemplo sería el siguiente:
  - **k=**3
- **numDenunciasPorTipo:** Número entero de archivos de cada tipo con los que realizar la ejecución. Debe ser un entero menor o igual a 80. Si establecemos este parámetro, por ejemplo, a 75. El sistema cogerá las 75 primeras denuncias de cada tipo.
  - **numDenunciasPorTipo=**75

Si queremos contar con la totalidad del corpus, 110 denuncias de Odio y 80 de No Odio, deberemos introducir el siguiente parámetro:

  - **numDenunciasPorTipo=**-1
- **procesamiento:** Valor entero con el que elegir el procesamiento del texto. Comandos para cada procesamiento:
  - Procesamiento 1:
    - **procesamiento=**1
  - Procesamiento 2:
    - **procesamiento=**2
  - Procesamiento 3:
    - **procesamiento=**3
  - Procesamiento 4a:
    - **procesamiento=**4
  - Procesamiento 4b:
    - **procesamiento=**5

¡Importante! Cabe destacar que este parámetro en el modo entrenamiento debe ser 2, ya que los textos son tratados únicamente con el procesamiento 2 en el controlador web. Fue elegido al tratarse del procesamiento con mejores resultados.

- **semilla:** Entero con la semilla para la ejecución.
  - Por ejemplo: **semilla=**42

9.2. Manual de usuario de la web

Realizaremos la ejecución del fichero *.py controladorPagina* mediante comando o desde un entorno de desarrollo Python como PyCharm. Página a la que podremos acceder en nuestro navegador en la dirección http://127.0.0.1:5000/. Inicialmente nos redirigirá a la vista principal:

Ilustración 97: Uso de Escribir Texto

Donde cada número corresponde a:

- 1: Botón para volver a la vista principal.
- 2: Botón para elegir si queremos subir un archivo o escribir un archivo.
- 3: Caja de selección con el modelo que queremos usar para la clasificación.
- 4: Caja donde escribiremos la denuncia a clasificar. Se recomienda copiar el texto de una denuncia existente en el corpus.
- 5: Botón para clasificar el texto de la denuncia.

Si pulsamos en escribir un archivo, se nos redirigirá a la siguiente vista:

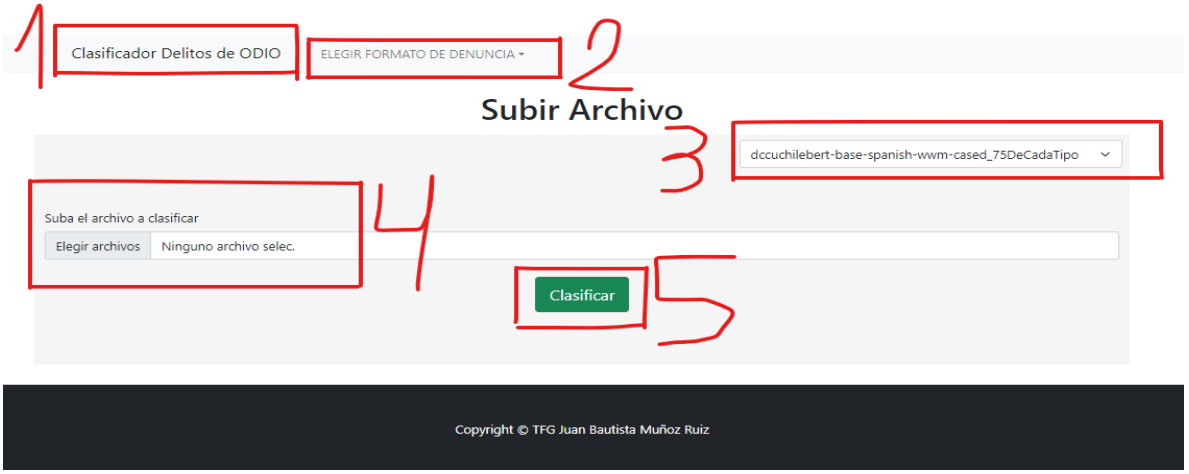

Ilustración 98: Uso de Subir Archivo

Donde cada número corresponde a:

- 1: Botón para volver a la vista principal.
- 2: Botón para elegir si queremos subir un archivo o escribir un archivo.
- 3: Caja de selección con el modelo que queremos usar para la clasificación.

- 4: Formulario para subir un archivo de texto. Una vez pulsado se nos abrirá el visualizador de carpetas de nuestro sistema operativo y bastará con situarse en la carpeta correcta y clickar en subir archivo.
- 5: Botón para clasificar la denuncia.

Más información sobre los mensajes de error y de visualización en el punto 5.7.

9.2.1. Modelos Disponibles en la web

A continuación se detallan todos los modelos presentes en la aplicación web para su correspondiente prueba y visualización de resultados.

Contamos con un corpus de 110 denuncias de Odio y 80 de No Odio. Los modelos fueron entrenados con determinados fragmentos del corpus explicados a continuación.

- **dccuchilebert-base-spanish-wwm-cased_75DeCadaTipo**
  - Se trata de un modelo BETO sobre el que se ha realizado el ajuste fino con una parte del corpus de denuncias.
  - Ha sido entrenado con las primeras 75 denuncias de Odio y 75 de No Odio. Por lo que las 35 últimas denuncias de Odio y las 5 últimas de Odio están reservadas para la prueba y visualización de resultados mediante subida de escritura o archivos a la web.
- **distilbert-base-uncased_75deCada:**
  - Este es un modelo BERT en su versión reducida con palabras en minúscula.
  - Al igual que el anterior, ha sido entrenado con las primeras 75 denuncias de Odio y 75 de No Odio. Ergo cuenta con los mismos archivos reservados para la prueba
- **Naive-75deCada:**
  - Es un modelo Naive Bayes de Sk-learn entrenado con una parte del corpus.
  - Como su propio nombre indica, y como los dos anteriores, ha sido entrenado con las primeras 75 denuncias de Odio y 75 de No Odio. Para los archivos de prueba contamos con la misma casuística descrita anteriormente.
- **PlanTL-GOB-ESroberta-base-bne_CorpusSubcojuntoKFold**
  - Se trata de un modelo BERTIN sobre el que se ha realizado el ajuste fino con el corpus.
  - Sin embargo, este modelo ha sido entrenado de la misma manera que en la última división k-fold en la validación cruzada con la semilla 42. Es decir , una división con *k=3* donde se divide el corpus en tres subconjuntos. Dos se usan para el entrenamiento, 127 denuncias, y una para la validación, 63 denuncias.

        - Ha sido entrenado con 127 denuncias.Todo ello siguiendo la distribución original de etiquetas. Es decir, 55 denuncias de No Odio y 72 de Odio.
        - Para la validación de resultados se reservan un total de 63. Es decir, un tercio del corpus. Estos archivos están situados en la carpeta *PruebaValidación*. Esta división sigue la distribución de clase del propio corpus. Como consecuencia de ello, encontramos 38 de Odio y 25 de No Odio para la prueba manual.
- **SVM-75deCada**
    - Se trata de un modelo SVM de Sk-learn entrenado con una parte del corpus.
    - Siguiendo el mismo método que en los tres primeros, ha sido entrenado con las primeras 75 denuncias de Odio y 75 de No Odio. Para los archivos de prueba contamos con el mismo caso descrito en los primeros 3 modelos.

10. CONCLUSIONES

El sistema desarrollado tiene alto rendimiento en la predicción de la tipología de las denuncias policiales por lo que cumple con creces el requisito fundamental. Además, la interfaz web es bastante intuitiva, web-responsive y fácil de utilizar.

El estudio previo realizado abarcando varias tecnologías existentes ha ayudado a la compresión del ámbito del problema y su resolución. Se han aprendido conceptos nuevos y afianzado otros muchos otros. En adición, se ha ampliado la visión general sobre el espectro de campos apasionantes y extensos como el Aprendizaje Automático, Procesamiento del Lenguaje Natural, Ciencia de Datos e Inteligencia Artificial.

Por otro lado, como aspectos futuros a mejorar numerosos y de alta variedad los que podrían tener en cuenta. Desde acceso desde un servidor web online, una mayor complejidad de los servicios web ofrecidos o el uso de técnicas más complejas para el procesamiento y análisis de los resultados obtenidos.

En líneas generales, este Proyecto de Fin de Grado ha sido bastante entretenido de desarrollar y muy compaginable, en términos de contenidos y tecnologías, con otras asignaturas en curso y cursadas anteriormente.

Bibliografía

Cámara, M. E. (2023). Prácticas de la asignatura Procesamiento del Lenguaje Natural. Prácticas 2-7 y Proyecto Final. Universidad de Jaén

Dot. (2017, 16 de diciembre). Regresión Lineal y mínimos Cuadrados. https://www.youtube.com/watch?v=k964_uNn3l0&t=443s

Dot. (2018a, 19 de marzo). ¿Qué es una Red Neuronal? Parte 1: La Neurona. https://www.youtube.com/watch?v=MRIv2IwFTPg

Dot. (2018b, 28 de mayo). ¿Qué es una Red Neuronal? Parte 2: La Red. https://www.youtube.com/watch?v=uwbHOpp9xkc&t=376s

Dot. (2018c, 3 de octubre). ¿Qué es una Red Neuronal? Parte 3: Backpropagation. https://www.youtube.com/watch?v=eNIqz_noix8&t=359s

Dot. (2021, 14 de noviembre). ¿Por qué estas Redes Neuronales son tan Potentes?.Parte 2 https://www.youtube.com/watch?v=xi94v_jl26U

Ganesan, K. (2020). HashingVectorizer vs CountVectorizer. Kavita Ganesan. https://kavita-ganesan.com/hashingvectorizer-vs-countvectorizer/

García, J. D. (2020, 23 de octubre). Redes Neuronales desde cero (I). IArtificial.net. https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/

Gautam, H. (2021, 13 de diciembre). Word Embedding Basics. Medium. https://medium.com/@hari4om/word-embedding-d816f643140

Github. (s.fa.). BETO: BERT Español. https://github.com/dccuchile/beto

Github. (s.fb.). Spanish Language Models. https://github.com/PlanTL-GOB-ES/lm-spanish

Heras, J. M. (2020a, 2 de octubre). Regresión Lineal: teoría y ejemplos en Python. IArtificial.net. https://www.iartificial.net/regresion-lineal-con-ejemplos-en-python/

Heras, J. M. (2020b, 9 de octubre). Precision, Recall, F1, Accuracy en clasificación. IArtificial.net. https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/

Huggin Face. (s.fa.). Bert Base Model Uncased. https://huggingface.co/bert-base-uncased

Huggin Face. (s.fb.). Bert Base Model Cased. https://huggingface.co/bert-base-cased

Huggin Face. (s.fc.). Distil-Bert. https://huggingface.co/docs/transformers/model_doc/distilbert

Huggin Face. (s.fd.). Bertin. https://huggingface.co/bertin-project/bertin-roberta-base-spanish

Huggin Face. (s.fe.). Multilingual-Bert. https://huggingface.co/bert-base-multilingual-cased

Huggin Face. (s.ff.). Text Classification. https://huggingface.co/docs/transformers/v4.24.0/en/tasks/sequence_classification

Ministerio de Interior. (2022, 3 de octubre). Las Fuerzas de Seguridad investigaron en 2021 un total de 1802 delitos de odio. https://www.interior.gob.es/opencms/es/detalle/articulo/Las-Fuerzas-de-Seguridad-investigaron-en-2021-un-total-de-1.802-delitos-de-odio/

Monica. (2022, 11 de octubre). Word2Vec: CBOW Vs Skip-Gram. Medium. https://medium.com/mlearning-ai/word2vec-cbow-and-skip-gram-55d23e64d8b6

Murzone, F. (2022, 30 de marzo). Extracción de Features con Bag of Words (BOW) y TF-IDF para NLP. Medium. https://medium.com/escueladeinteligenciaartificial/extracci%C3%B3n-de-features-con-bag-of-words-bow-y-tf-idf-para-nlp-f89d678abc0e

Na. (2022, 8 de noviembre). ¿Cómo funcionan los Transformers en Español?. Aprende Machine Learning. https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/

Nabil, M. (2023, 19 de abril). Unpacking the Query, Key and Value of Transformers: An Analogy to Database Operations. LinkedIn. https://www.linkedin.com/pulse/unpacking-query-key-value-transformers-analogy-database-mohamed-nabil/

NBro. (2020, 21 de septiembre). How is BERT different from the original transformers architecture?. Stackexchange Https://ai.stackexchange.com/questions/23221/how-is-bert-different-from-the-original-transformer-architecture

IONOS. (2022, 2 de noviembre). Flask vs Django: una comparativa de los frameworks de Python. IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/flask-vs-django/

Pai, A. (2023, 9 de mayo). Analyzing 3 Types of Neural Networks in Deep Learning. Analytics Vidhya https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Patel, H. (2019). Text classification using K Nearest Neighbors (KNN). Opengenus.org. https://iq.opengenus.org/text-classification-using-k-nearest-neighbors/

Paul, S. (2021, 10 de diciembre). A detailed case study on Multi-Label Classification with Machine Learning algorithms and predicting movie tags based on plot summaries!. Medium. https://medium.com/@saugata.paul1010/a-detailed-case-study-on-multi-label-classification-with-machine-learning-algorithms-and-72031742c9aa

Rodriguez, C. C. (2021, 15 de diciembre). Máquina de Soporte Vectorial. Medium. https://medium.com/@csarchiquerodriguez/maquina-de-soporte-vectorial-svm-92e9f1b1b1ac

Requena, A et al. (d.f.). Nuevas Tecnologías y Contaminación de Atmósferas para PYMEs. Universidad de Murcia. https://www.um.es/LEQ/Atmosferas/Ch-VI-3/C63s6p2.htm

Schmid, P. (2020, 7 de abril). K-Fold as Cross-Validation with a BERT Text-Classification Example. Philschmid. https://www.philschmid.de/k-fold-as-cross-validation-with-a-bert-text-classification-example

Sefidian, A. M. (2023, 16 de abril). Understanding micro, macro and weighted averages for SciKIT-Learn metrics in multi-class classification with example. Sefidian Academy. http://iamirmasoud.com/2022/06/19/understanding-micro-macro-and-weighted-averages-for-scikit-learn-metrics-in-multi-class-classification-with-example/

Stecanella, B. (2017, 25 de mayo). A practical explanation of a Naive Bayes Classifier. MonkeyLearn. https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/

Sk Learn. (s.f.). Sk Learn Metrics. Precision, Recall, FScore and Support. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

Universidad de Oviedo. (d.f). El Algoritmo k-means aplicado a clasificación y procesamiento de imágenes. https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html

Universidad Europea. (2022, 7 de julio). Aprendizaje supervisado y no supervisado. https://universidadeuropea.com/blog/aprendizaje-supervisado-no-supervisado/

Ureña, L. A. (s.fa). Temario de la asignatura Procesamiento del Lenguaje Natural, Tema 3, Aplicaciones. (pp. 3-7). Universidad de Jaén.

Ureña, L. A. (s.fb). Temario de la asignatura Procesamiento del Lenguaje Natural, Tema 3, Aplicaciones. (pp. 50-51). Universidad de Jaén.

Ureña, L. A. (s.fc). Temario de la asignatura Procesamiento del Lenguaje Natural, Tema 2, Recursos y Herramientas Lingüísticas. (pp. 18-19). Universidad de Jaén.

Ureña, L. A. (s.fd). Temario de la asignatura Procesamiento del Lenguaje Natural, Tema 2, Recursos y Herramientas Lingüísticas. (pp. 46-50). Universidad de Jaén.

Wikipedia. (2023a). Perceptrón. https://es.wikipedia.org/wiki/Perceptr%C3%B3n

Wikipedia. (2023b). Sigmoid function. https://en.wikipedia.org/wiki/Sigmoid_function

Wikipedia. (2023c). Validación Cruzada. https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada

Yoseo. (2017, agosto). La lista definitiva de stop words en español. https://www.yoseomarketing.com/blog/stop-words-en-espanol-lista-definitiva/