Universitat Oberta de Catalunya

uoc.edu

Análisis de sentimiento de textos basado en opiniones de películas usando algoritmos de aprendizaje computacional

Jorge Chulilla Alcalde
Grado de Informática
Inteligencia Artificial

David Isern Alarcón
Carles Ventura Royo

08/06/2021

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

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

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

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

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

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

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

B) GNU Free Documentation License (GNU FDL)

Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis de sentimiento de textos basado en opiniones de películas usando algoritmos de aprendizaje computacional*
Nombre del autor:	*Jorge Chulilla Alcalde*
Nombre del consultor/a:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Análisis de sentimiento, Machine Learning, Procesamiento del Lenguaje Natural*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Dentro del campo del Procesamiento de Lenguaje Natural (NLP) se ha generado un ecosistema de algoritmos que permiten el análisis y clasificación de datos a partir del aprendizaje automatizado.

Este trabajo se focaliza en el llamado análisis de sentimiento, por el cual, es posible clasificar textos en idioma inglés según una polaridad, en nuestro caso positiva o negativa. Como caso de uso concreto nos hemos decidido por textos de opiniones de películas o series en IMDB, considerando la importancia que el sector audiovisual tiene en la actualidad y la gran cantidad de recursos que destina esta industria al análisis de las preferencias de los usuarios.

Se ha definido y seguido una planificación standard de tareas a tres meses vista, utilizando una metodología de tipo *CRISP-DM*, que es la utilizada para proyectos basados en *"Machine Learning".*

Dentro del análisis inicial y después de ver las diferentes técnicas y algoritmos dedicados al NLP se han implementado y probado tres algoritmos diferentes, basados en conceptos distintos: por un lado, los algoritmos más clásicos, Multinomial Naïve Bayes y Logistic Regression, y por otro lado ULMFiT, basado en técnicas de *"Transfer Learning"*.

Finalmente, comprobamos que los resultados han sido muy buenos con unas tasas en los tres casos entorno al 90%, siendo ULMFiT el que mejores resultados obtiene. En este sentido, los recursos necesarios para este tipo de algoritmos pueden no justificar su utilización, considerando que la diferencia de resultados no ha sido demasiado grande, pero sí plasma su potencial.

Abstract (in English, 250 words or less):

Within the field of Natural Language Processing (NLP) an ecosystem of algorithms has been generated, allowing data analysis and classification using machine learning techniques.

This work is focused on the so-called sentiment analysis, which allows to classify texts in the English language according to a polarity, in our case positive or negative. As a specific use case, we have decided on texts of opinions of movies or series using IMDB, considering the importance that the audiovisual sector has and the large amount of resources that this industry allocates for user preferences analysis.

A standard three-month planning of tasks has been defined and followed, using a CRISP-DM type methodology, which is used for projects based on “Machine Learning”.

Within the initial analysis and after seeing the different techniques and algorithms dedicated to NLP, three different algorithms have been developed and tested, based on different concepts: firstly, the more classical algorithms, Multinomial Naïve Bayes and Logistic Regression, and finally ULMFiT, based on “Transfer Learning” techniques.

Finally, the results have been very good, with accuracies of around 90% in all three cases, being ULMFiT the best of all. In this sense, the resources necessary for this type of algorithm may not justify its use, considering that the differences has not been too big, but it does reflect its potential.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Actualmente la industria del entretenimiento es una de las que más volumen de negocio factura a nivel mundial, apoyándose en una situación social favorable que la beneficia. Tradicionalmente el sector audiovisual focalizado en cine y televisión es uno de sus grandes pilares, si bien es cierto que hay otros jugadores que reclaman cuota de mercado como son el sector de los videojuegos y los *eSports*.

Todo y que la situación actual de pandemia mundial ha afectado la producción de nuevas películas y series, por el contrario, ha sido un buen momento para aumentar clientes en plataformas de *streaming* que permiten una experiencia similar a la de las salas de cine, pero con la seguridad del hogar propio. Según datos publicados por la consultora PwC[1] [1] la suscripción a distintas plataformas ya preveía un incremento de 13,5$ billones a 24,5$ billones en el periodo que va del 2019 a 2024. Estos datos pueden verse afectados a corto plazo debido, como comentamos, a la situación actual mundial, pero se verá beneficiado a largo plazo con la aparición de nuevas plataformas, la integración de más países produciendo contenidos y la adquisición de nuevos derechos de propiedad.

Hoy en día la maximización de venta de los productos está fuertemente ligada al conocimiento y los datos relacionados. Con la situación que hemos planteado, sobre todo, las plataformas online tienen un arma muy avanzada para recoger información de sus contenidos y determinar si el producto gusta o no a sus clientes. Si a esto se añade la posibilidad de tener esta información por otros canales, como pueden ser redes sociales, canales dedicados, etc., la explotación de estos datos sólo está limitada a la aplicación de una inteligencia que saque información relevante de la misma.

Es por esta razón por la cual la clasificación de sentimientos es muy importante para este sector, ya que permite tomar decisiones estratégicas que mejoren las capacidades actuales y a futuro de su producto, adaptando las producciones a los gustos del espectador o maximizando aquellas sin tanta visibilidad inicial, entre tantas otras opciones.

Sin embargo, el disponer de esta información tan básica no es tan sencillo si sólo se dispone a priori de opiniones utilizando texto libre, y considerando también la posible jerga particular de cada persona o localización. Es en este

[1] https://www.pwc.com/us/en/industries/tmt/library/global-entertainment-media-outlook.html

punto donde el aprendizaje automatizado y la aplicación de algoritmos de *machine learning* (ML) aplicada a textos toman relevancia.

1.2 Objetivos del Trabajo

Dada la definición del trabajo y el tiempo disponible hemos considerado definir dos tipos de objetivos: uno principal que se centrará en la parte IA (Inteligencia Artificial) exclusivamente y que será el núcleo mínimo exigible, y otro final, por el cual se implementa un producto que permita al usuario evaluar las bondades del objetivo principal.

1.2.1 Objetivo Principal

El objetivo principal del proyecto es el análisis y evaluación de pruebas de algoritmos clasificadores de sentimientos dedicados al análisis de textos relacionados con opiniones de películas o series, para la obtención finalmente de un único clasificador que aúne los mejores resultados. El sentimiento se dividirá en las categorías positivo y negativo. En esta tarea se realizará un trabajo de prueba-error modificando las parametrizaciones de los algoritmos para obtener los mejores resultados utilizando técnicas de evaluación de modelos (tasa de errores, precisión o exactitud entre otras).

Revisando los posibles *datasets* disponibles consideramos que el analizador de textos se fundamentará en idioma inglés, ya que, será uno de estos repositorios la base de datos que la aplicación utilizará para el autoaprendizaje.

Dentro del alcance para este objetivo se incluye:

- Análisis de sistemas actuales NLP (*"Natural Language Processing"*) y algoritmos de aprendizaje computacional.
- Desarrollo del sistema NLP.
  - Definición del *dataset*. Búsqueda de repositorios de información que nos ayude en la fase de aprendizaje de los algoritmos.
  - Preparación de los datos. Una vez se tenga la información base se realizará una depuración y selección.
  - Modelado. Aplicación de los algoritmos. Realización de pruebas de rendimiento para optimización de los modelos.
  - Evaluación de los resultados.

Por el contrario, quedaría fuera del alcance:

- Carga automatizada de la información. Se definirá un *dataset* que nos ayude en la generación de los modelos de aprendizaje.
- La generación de modelos de ML automatizado. Sólo se generarán bajo demanda utilizando el software pertinente.

1.2.2 Objetivo Final

Por último, el objetivo final sería dar la posibilidad a cualquier usuario de probar el clasificador mediante texto libre. Dado que no es posible validar el contenido del texto, será responsabilidad del usuario ceñir su utilización al dominio tratado (opiniones de películas), ya que, en cualquier otro caso los resultados podrían no ser satisfactorios.

Para este objetivo se utilizará una aplicación *front-end* web de entrada de datos que nos retornará el resultado aplicando un algoritmo previamente generado. El desarrollo de esta aplicación también incluye el *deploy* de la misma para su accesibilidad vía internet.

Dentro del alcance para este objetivo se incluiría lo siguiente:

- El desarrollo y generación de un aplicativo *front-end* web para usuario.
- El *deploy* del aplicativo en una plataforma *cloud* para su acceso vía web.

Por otro lado, quedaría fuera del alcance:

- La representación gráfica o *reporting* de la información obtenida. Toda información relevante será mostrada por la aplicación web.

1.3 Enfoque y método seguido

Para enfocar el proyecto y establecer unas bases se ha buscado y revisado artículos de investigación académicas y revistas científicas relacionadas. De esta manera hemos tratado de evitar repeticiones en la medida de lo posible, todo y que ya existen multitud de trabajos relacionados utilizando datos y técnicas variadas.

La estrategia principal se fundamenta en la generación de un software totalmente nuevo y adaptado a la necesidad del proyecto. Para ello el *core* principal será el software Python desarrollado y que contendrá el código y las librerías específicas para la generación de modelos ML. En un segundo nivel, menos prioritario, enfocaremos la parte de desarrollo de la aplicación web y su despliegue.

Por su importancia y complejidad creemos que la parte *back-end Python* debe ser la que primero se desarrolle, ya que nos permitirá probar diferentes alternativas y nos dejará margen posible de maniobra en caso de que haya desajustes en los tiempos de planificación y la entrega de hitos. Para la correcta entrega de este apartado será imprescindible la búsqueda de trabajos y artículos publicados que nos pueda ayudar a poner las bases adecuadas. De entrada, nos planteamos los siguientes puntos como básicos a la hora de afrontar este módulo:

- Búsqueda de un *dataset* que nos ayude en la construcción del modelo.

- Preparación de los datos del *dataset*.
- Desarrollo de la aplicación Python obteniendo los modelos.
- Evaluación de los modelos seleccionados.
- Obtención del código final y de los modelos parametrizados seleccionando aquellos que arrojen los mejores resultados.

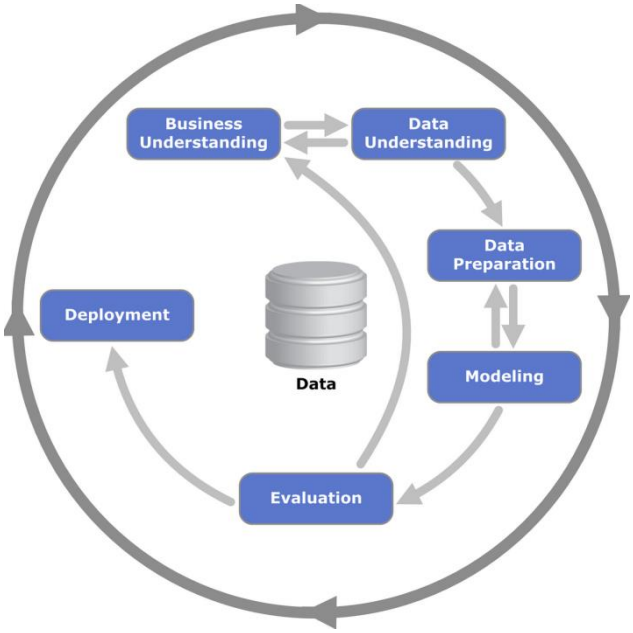


Figura 1. Diagrama de proceso basado en CRISP-DM

Como se puede apreciar este apartado puede ser recursivo en cuanto a su progresión, ya que, es muy probable que encontremos sobre la marcha información que nos pueda ayudar a la obtención de mejores resultados, por lo que entraríamos en un estado de prueba y error en búsqueda de una mejora continua en el producto final.

Los proyectos de este tipo se basan en el procesamiento del lenguaje natural (*"Natural Language Processing")* o NLP, por el cual se interpreta y procesa el lenguaje humano por máquinas y algoritmos. Actualmente se utiliza una metodología standard de desarrollo en la que hay una serie de fases a resolver de manera secuencial. Se trata del modelo CRISP-DM[2] [2] [3] (figuras 1 y 2). Esta metodología será la que utilizaremos en el desarrollo de la aplicación Python y será explicada con más detalle posteriormente.

[2] https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome

https://towardsdatascience.com/the-beginning-of-natural-language-processing-74cce2545676

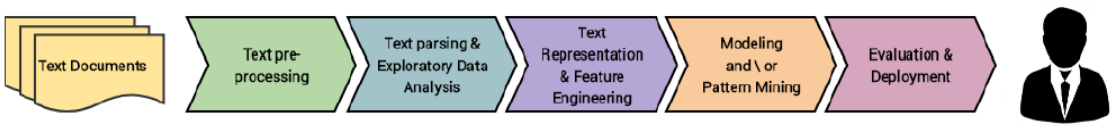


Figura 2. Workflow basado en CRISP-DM

Por último, el bloque de desarrollo, prueba y despliegue web del aplicativo tiene como objetivo la prueba de resultados obtenidos en el bloque anterior, pero no debe ser el objetivo principal del proyecto. Por esta razón parte del tiempo dedicado puede ser sacrificado en beneficio del bloque *back-end* en caso de necesidad. En cualquier caso, se respetarán los métodos de desarrollo típicos en este tipo de utilidades y que se resumen en:

- Diseño de la parte visual y del *look & feel* de la aplicación.
- Desarrollo del código relacionado.
- Pruebas del aplicativo.
- Despliegue del aplicativo para su uso general.

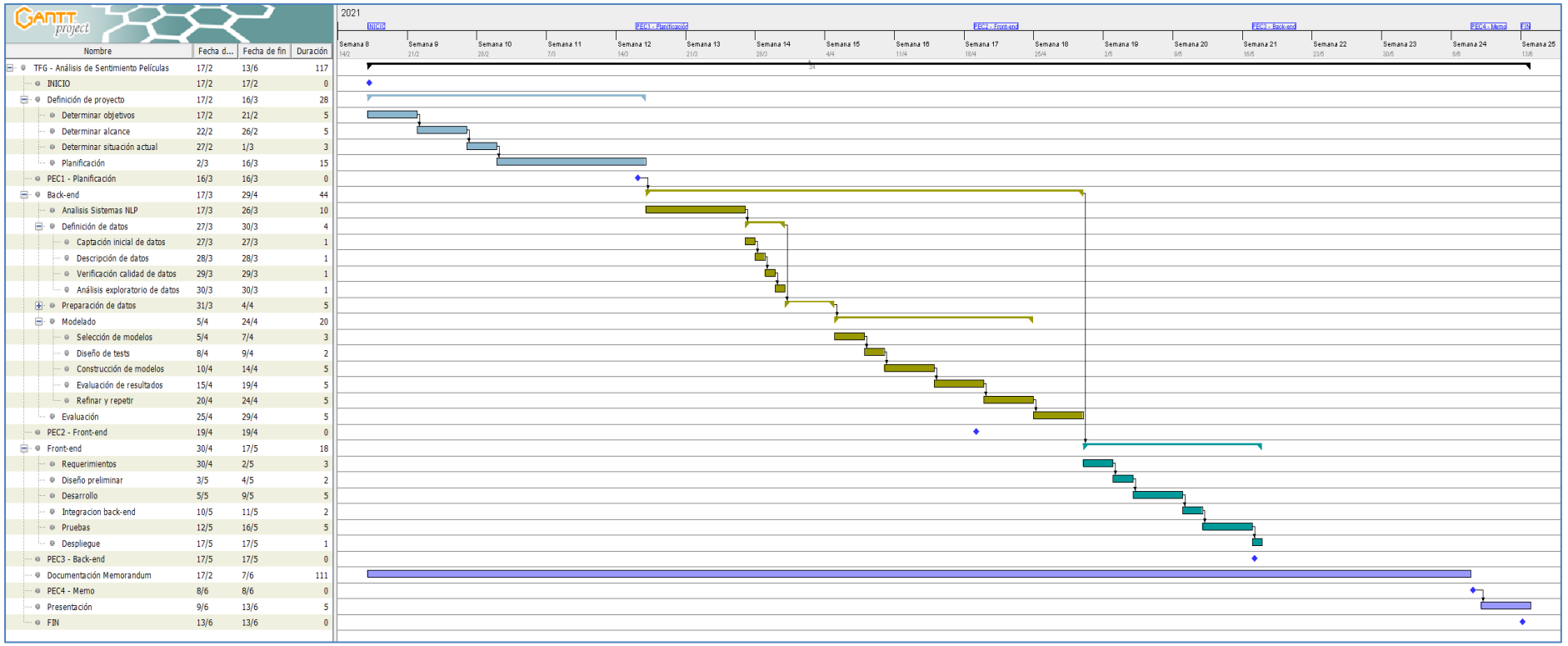
Como requisito fundamental en este bloque tenemos que los modelos ML deben existir previamente para que puedan invocarse por la utilidad.

1.4 Planificación del Trabajo

Incluimos a continuación un cuadro con los puntos de entrega detallados (figura 3) y los hitos por entrega de prueba continua, así como la planificación en un diagrama de Gantt (figura 4). En esta planificación se abarca desde el periodo inicial de apertura de TFG en la que se define el proyecto, hasta la fecha final de entrega de la presentación de la defensa.

Fase	Descripción tarea	Fecha prevista	Duración
Definición de proyecto	**INICIO**	**17-Febrero**	
	Determinar objetivos	21-Febrero	5d
	Determinar alcance	26-Febrero	5d
	Determinar situación actual	1-Marzo	3d
	Planificación	16-Marzo	15d
	Hito 1	**16-Marzo**	
Back-end	Análisis sistemas NLP	26-Marzo	10d
	Definición de datos	30-Marzo	4d
	Preparación de datos	4-Abril	5d
	Modelado (evaluación de resultados)	19-Abril	15d
	Hito 2	**19-Abril**	
	Refinar y repetir	24-Abril	5d
	Evaluación	29-Abril	5d
Front-end	Requerimientos	2-Mayo	3d
	Diseño preliminar	4-Mayo	2d
	Desarrollo	9-Mayo	5d
	Integración back-end	11-Mayo	2d
	Pruebas	16-Mayo	5d
	Despliegue	17-Mayo	1d
	Hito 3	**17-Mayo**	
Memorándum	Documentación memorándum	8-Junio	22d
	Hito 4	**8-Junio**	
Presentación	Documentación presentación	13-Junio	5d
	FIN	**13-Junio**	

Figura 3. Tabla cronológica de la planificación



Nombre	Fecha d...	Fecha de fin	Duración
TFG - Análisis de Sentimiento Películas	17/2	13/6	117
INICIO	17/2	17/2	0
Definición de proyecto	17/2	16/3	28
Determinar objetivos	17/2	21/2	5
Determinar alcance	22/2	26/2	5
Determinar situación actual	27/2	1/3	3
Planificación	2/3	16/3	15
PEC1 - Planificación	16/3	16/3	0
Back-end	17/3	29/4	44
Analisis Sistemas NLP	17/3	26/3	10
Definición de datos	27/3	30/3	4
Captación inicial de datos	27/3	27/3	1
Descripción de datos	28/3	28/3	1
Verificación calidad de datos	29/3	29/3	1
Análisis exploratorio de datos	30/3	30/3	1
Preparación de datos	31/3	4/4	5
Modelado	5/4	24/4	20
Selección de modelos	5/4	7/4	3
Diseño de tests	8/4	9/4	2
Construcción de modelos	10/4	14/4	5
Evaluación de resultados	15/4	19/4	5
Refinar y repetir	20/4	24/4	5
Evaluación	25/4	29/4	5
PEC2 - Front-end	19/4	19/4	0
Front-end	30/4	17/5	18
Requerimientos	30/4	2/5	3
Diseño preliminar	3/5	4/5	2
Desarrollo	5/5	9/5	5
Integracion back-end	10/5	11/5	2
Pruebas	12/5	16/5	5
Despliegue	17/5	17/5	1
PEC3 - Back-end	17/5	17/5	0
Documentación Memorandum	17/2	7/6	111
PEC4 - Memo	8/6	8/6	0
Presentación	9/6	13/6	5
FIN	13/6	13/6	0

Figura 4. Diagrama Gantt de la planificación

Considerando, sobre todo, las problemáticas en la aplicación de algoritmos de aprendizaje automatizado consideramos como factores de riesgo:

- La obtención de un modelo de ML con un tiempo de proceso adecuado. De todos es conocida la demanda de tiempo de máquina que algunos algoritmos necesitan. En principio no se considera utilizar grandes sistemas de procesado como pueden ser aplicaciones *cloud*, por lo que será fundamental ajustar los algoritmos en base a las capacidades de los recursos disponibles.
- La obtención de resultados óptimos en las predicciones de los algoritmos. No es admisible por ejemplo que un determinado algoritmo arroje una precisión del 15%.
- Encontrar un *dataset* o repositorio de datos con información suficiente y de calidad para el entrenamiento del algoritmo de ML.
- Entrega del objetivo final, aplicación web para usuario, ajustándose a la planificación. Se recuerda de todas formas que el objetivo principal se centra en la parte *back-end*.

Otros factores de riesgo derivados serían:

- Finalización de los objetivos dispuestos en tiempo y forma. Será necesario en este punto hacer un balance correcto de las prioridades y un ajuste estricto al alcance del proyecto. En última instancia sería necesario revisar el alcance del proyecto para ajustar el resultado al calendario.
- Alineamiento del avance del proyecto a las expectativas. Será necesario la realización de un seguimiento de las tareas y su evolución periódica. El *feedback* del avance será fundamental en este sentido, permitiendo un reajuste ágil de tareas y desarrollos.

1.5 Breve sumario de productos obtenidos

Los productos que serán entregados al final del proyecto serán:

- **Código Python**. El código fuente utilizado para la obtención de los modelos ML se facilitará en un fichero comprimido.
- **Código parte web**. El código fuente de la parte *front-end* se facilitará en un fichero comprimido.
- **Aplicación web de prueba**. Se facilitará una aplicación web para la prueba de los resultados obtenidos.
- **Memorándum del proyecto**. La documentación explicativa del proyecto se facilitará en formato PDF.

El código fuente resultante se compartirá en el fichero de entrega del TFG con un archivo zip comprimido.

Adicionalmente se facilitará una dirección web en GitHub para compartir todos los ficheros considerados relacionados con el desarrollo realizado (código fuente, modelos NLP, ficheros de configuración, etc.).

1.6 Breve descripción de los otros capítulos de la memoria

Capítulo 2. Análisis de sistemas actuales NLP y algoritmos de aprendizaje computacional. Realización de un estudio teórico en el cual se pueda dictaminar los algoritmos a utilizar en la fase de desarrollo. El estudio se basará en la revisión de trabajos anteriores publicados, así como de la propia experiencia personal y profesional. Se orientará especialmente al análisis de sentimiento o la minería, definiendo las problemáticas asociadas como, por ejemplo, el uso de spam o la detección de ironía o sarcasmo. El núcleo de este apartado será el relacionado con las técnicas utilizadas, desde la preparación previa de los datos, pasando por la selección y extracción de características a la generación de los modelos, desde los más clásicos hasta las últimas tendencias.

Capítulo 3. Desarrollo del sistema NLP. A partir del análisis realizado comenzaremos la aplicación del sistema para el cual afrontaremos distintas fases: definición del *dataset*, preparación de los datos, modelado y evaluación de los resultados.

Capítulo 4. Desarrollo del sistema *front-end*. Una vez desarrollada la parte *back-end* procederemos a desarrollar la *interface* con la cual se podrá probar los resultados obtenidos en un entorno web de usuario en un proceso de integración.

Capítulo 5. Conclusiones. Al final del trabajo procederemos a realizar un análisis y explicación de los resultados, así como otros puntos destacables como las lecciones aprendidas, puntos de mejora y posibles líneas de trabajo que se puedan derivar a futuro.

2. Análisis de sistemas NLP aplicados al análisis de sentimientos

2.1 Introducción

En los últimos años la problemática de la minería de datos aplicada a textos (en inglés *text mining*) ha experimentado una revolución, propiciada por la aparición de ingentes cantidades de información gestionadas por sistemas de "social media" en la web. Surge, por tanto, una necesidad de analizar esta información en formato no estructurado en beneficio científico, pero sobre todo en el ámbito privado con fines comerciales. Esto ha provocado la aparición de métodos y algoritmos que son capaces de procesar esta información e incluso focalizarlas según el ámbito a aplicar.

Precisamente uno de los problemas en estos tipos de sistemas es que no son capaces de tener una aplicación genérica para cualquier tipo de texto. Es por ello que el análisis semántico se realiza en base al ámbito de aplicación, construyendo diccionarios específicos que ayuden a construir un significado comprensible a partir de textos sin un orden ni estructura predefinidos.

Una de las potenciales utilidades del análisis de textos es la posibilidad de extraer la llamada "información de sentimiento", por la cual el ser humano expresa un tono emocional sobre algo. Esto derivaría en la posibilidad de etiquetar nuestras opiniones en una polaridad de tipo positiva, negativa o neutra ("ese gato está gordo", "este coche me gusta"). Este análisis del lenguaje humano tiene multitud de casos de uso en la actualidad con beneficios demostrados: aplicaciones de bolsa que cotejan los cambios de sentimiento en redes sociales, campañas de marketing, campañas electorales, recomendación de películas, etc.

2.2 Análisis de sentimientos. Definiciones

Existen diferentes formas de definir el análisis de sentimientos, también referido como minería de opinión (en inglés *opinion mining*). Nos quedamos con esta definición (Liu B. & Zhang L., 2012) [4] por la cual: *"Sentiment analysis or opinion mining is the computational study of people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics and their attributes."*
[El análisis de sentimiento o minería de opinion es el estudio computacional de opiniones de personas, evaluaciones, actitudes y emociones respecto a entidades, individuos, asuntos, eventos, temas y sus atributos.]

En el ámbito de la computación según Liu B.,2012 [5] las opiniones pueden describirse como una "quintupla" formada por cinco conceptos: el objeto sobre el que se basa la opinión, la característica del objeto a la que se hace referencia, el sentimiento sobre esta característica, el opinador y el momento

en el tiempo en el que se hace la opinión. De conseguir estas “quintuplas” en el análisis de un texto habríamos conseguido pasar de una información sin estructura inicial a una estructurada por la cual nuestro sistema NLP podría actuar. Ejemplos relacionados con las opiniones de películas podrían ser:

{Robocop, entretenimiento, positivo, ademir1, 11-ene-2005}
{Avatar, 3D, positivo, object_52, 23-mar-2015}
{Birdman, humor, negativo, raul53#, 21-abr-2020}

Las opiniones pueden clasificarse en diferentes grupos. Una de las más utilizadas es la siguiente:

- Regulares. Las más comunmente conocidas como opiniones. Se podrían dividir en dos subgrupos:
    - Directas. Opinión directa sobre una entidad. Ej.: “este coche es bonito”.
    - Indirectas. Son opiniones dadas sobre una entidad o característica de la entidad de forma indirecta a través de sus efectos en otras entidades. Ej.: “después de la revisión mi coche funciona bien” (se trataría de un sentimiento positivo no para el coche, sino para la revisión del coche).
- Comparativas. Se trata de opiniones basadas en la comparación de dos o más entidades que comparten ciertas similitudes [6].

Otros conceptos importantes relacionados serian los de subjetividad, emoción y estado anímico (“mood” en inglés):

- Subjetividad. Según [7] una sentencia subjetiva puede expresar algún tipo de sensación personal o creencia, pero no necesariamente un sentimiento. En contraposición con una sentencia objetiva, la cual expresa una información basada en hechos y que podemos considerar como opinión o sentimiento.
- Emoción. Sería una forma de expresión de nuestros sentimientos o pensamientos subjetivos. Estarian muy relacionadas con los sentimientos, ya que la fuerza de una opinión estaría ligada a la intensidad de la emoción. Odio, venganza, tristeza, amor,…, serian ejemplos de emociones que influirian en el sentimiento sobre una entidad.
- Estado anímico. En estudios recientes [8] se considera este concepto como una mezcla de sentimientos y emociones que mueven al autor a escribir un comentario, observación, crítica, etc.

Los sistemas NLP y concretamente los dedicados a análisis de sentimiento tienen que considerar también el tipo de texto a analizar. Según un caso u otro

el sistema tendrá que ajustar su procesado. Básicamente tendríamos tres tipos posibles de orígenes de información a analizar:

- Documentos. Se considera en este caso que un documento refleja una opinión sobre una entidad o una característica. Es un caso en el cual se ha de realizar todo un proceso de desglose y clasificación de conceptos (en inglés *document-level sentiment classification* [9]).

- Frases. Se trataría de un nivel más detallado. En este caso no tenemos el problema del análisis a nivel de documento por el cual es bastante complicado que sólo haya una entidad a la que se haga referencia. Este tipo de análisis se centraría sobre todo en los aspectos de subjetividad y objetividad por las cuales se puede expresar una opinión. [10]

- Entidades. Finalmente, el nivel de más detalle es el que hace referencia a entidades o características de entidades. De hecho, en el contexto de películas ya tenemos trabajos realizados basándose en este tipo de análisis, como por ejemplo el realizado por Tun Thura Thet, Jin-Cheon Na y Christopher S.G. Khoo; 2010 [11].

2.3 Problemáticas asociadas

Con lo explicado hasta ahora ya se puede observar la dificultad del desarrollo de un sistema que clasifique sentimientos en textos no estructurados. No es la intención de este estudio, pero a continuación incluimos otras problemáticas asociadas con la intención de ver la magnitud del problema al que se enfrentan los investigadores:

- Minería de textos en multilenguaje. Esta problemática en buena lógica debería construir un diccionario de lenguaje asociada a cada idioma. Sin embargo, no es fácil, debido a las connotaciones particulares de cada idioma. Actualmente la solución más utilizada es utilizar máquinas para la traducción [12].

- Agrupación de referencias que hagan referencia a lo mismo. En el lenguaje común el ser humano hace referencia a conceptos que son iguales con distintas palabras. Por ejemplo, en el campo de la fotografía, en inglés, los conceptos “*photo*” y “*picture*” hacen referencia a lo mismo: “fotografía”. Sin embargo, los conceptos “*picture*” y “*movie*” son sinónimos en el ámbito del cine. Pese a que se ha hecho un gran esfuerzo en la definición de diccionarios que reflejen estos sinónimos, el hecho que haya ámbitos que hagan cambiar el significado de palabras clave complica sobremanera el análisis de textos.

- Detección de spam. La utilización de *bots* que, de forma automatizada, insertan texto en las redes sociales es un problema

hoy en día. Con estas prácticas se trata de influir en la opinión de la mayoría a través de opiniones interesadas para unos determinados fines. Estas opiniones son comúnmente conocidas como *fakes* y es una problemática ya introducida por Jindal y Liu en [13].

- Sarcasmo e ironía [14]. Se trata de la detección de textos con contenido irónico o sarcástico. Es una tarea realmente complicada ya que actualmente no hay una definición común para los investigadores.

2.4 Técnicas

Existen diferentes trabajos y estudios relacionados con NLP pero, desde el punto de vista del análisis de sentimiento, tenemos uno muy interesante realizado por Wala Medhat, Ahmed Hassan y Hoda Korashy (2014) [15]. De este estudio nos quedamos con una imagen (Fig.5) en la que se presenta las diferentes técnicas aplicadas para el análisis de sentimiento.

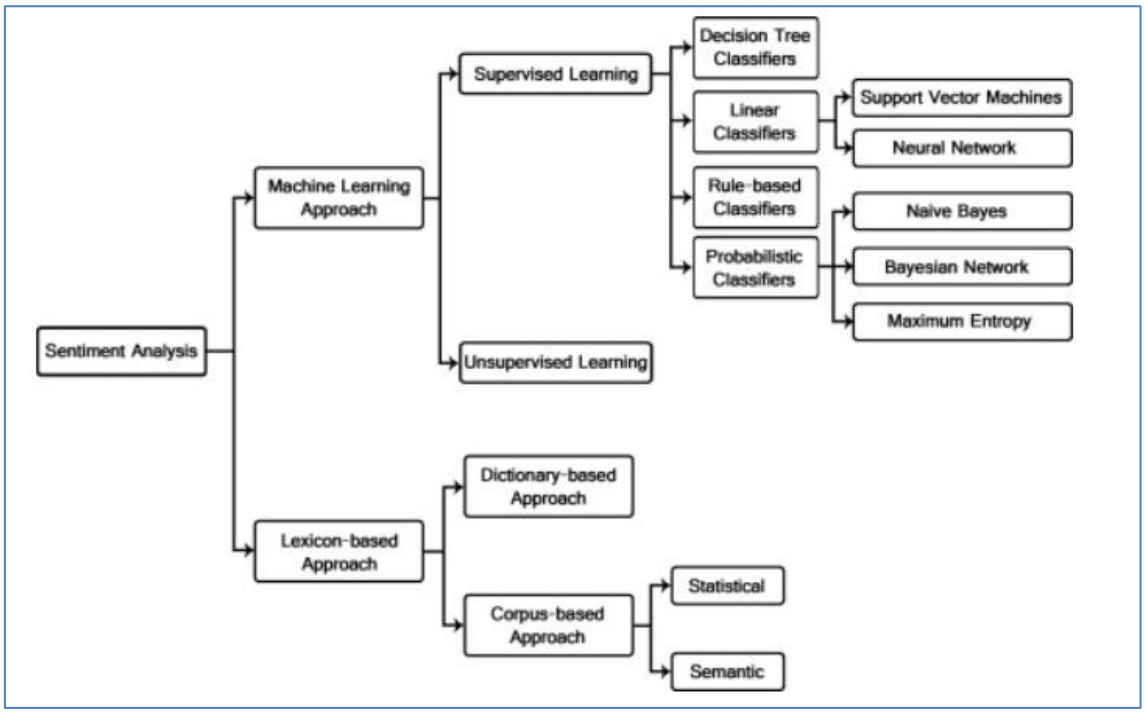


Figura 5. Clasificación de técnicas relacionadas con análisis de sentimiento.

Como se observa tendríamos dos grupos de técnicas: por un lado las basadas en la existencia de una base de datos léxica (en inglés *Lexicon-based Approach*), y por otro la que utiliza herramientas de ML (en inglés *Machine Learning Approach*). Otra técnica utilizada aunque menos debido a su complejidad sería una técnica híbrida, por la que se utilizarían las dos técnicas comentadas.

Sin embargo, previamente a la aplicación de herramientas de clasificación, es necesario realizar varios pasos previos. Para empezar seria necesario un

preprocesamiento de los datos, continuando con procesos de selección y extracción de características. Considerando que los datos de entrada no están lo suficientemente refinados para su estudio y clasificación, estas técnicas nos permiten reducir la complejidad que presentan. Problemas asociados que nos "obligan" a utilizar estas técnicas serían por ejemplo la existencia de datos redundantes o la existencia de "ruido", es decir, datos que son perjudiciales para nuestro sistema y que deben ser eliminados. De la misma forma la extracción de características nos puede permitir "enriquecer" nuestro conjunto de datos para un mejor y más óptimo análisis.

2.4.1 Preprocesamiento de datos

Existen diferentes aproximaciones a estas tareas, pero todas están siempre muy relacionadas al contexto de datos a tratar. En la red hay numerosos trabajos y técnicas utilizadas así como guías de preparación de datos [16] [17], pero el consenso común es la importancia de este punto para la obtención de resultados correctos, como por ejemplo se cita en un artículo de Erik Cambria, Poria Soujanya, Alexander Gelbukh y Mike Thelwall en 2017 [18].

Las técnicas más habituales las podemos ver a continuación.

2.4.1.1 Eliminación de ruido

Consiste en eliminar caracteres que interfieran en el análisis del texto. Es una de las técnicas más importantes y usadas de preprocesamiento. Ejemplos más comunes de datos a eliminar:

- *Links* html.
- Marcas de *html* ("*tags*").
- Expresiones entre corchetes.
- Caracteres no ASCII.
- Nombres de usuario de twitter.
- Espacios en blanco.
- Caracteres especiales como "\", "[", "]", etc.

2.4.1.2 StopWords

Se trataría de "eliminar" palabras que no aporten nada al análisis de la información, como por ejemplo artículos, pronombres,… Puede ser muy útil para mejorar los tiempos del algoritmo ya que reduce el vocabulario usado. En un artículo de conferencia publicado en 2011 [19] hemos encontrado un estudio de aplicación parecido a nuestro caso: analisis de sentimiento para *reviews* de películas. En ella se obtienen unos porcentajes de precisión mejorados aplicando *Stopwords* junto a un clasificador Naïve Bayes.

# topics (LDA)	accuracy (SVM) positive reviews	negative reviews
10	25%	23%
15	50%	35%
20	54%	43.5%
25	44%	30%

Table 2: Effect of content words on accuracy

# topics (LDA)	accuracy (SVM) positive reviews	negative reviews
10	60.5%	41%
15	62.5%	53%
20	46.5%	35%
25	78.5%	72%

Table 3: Effect of stop words and gaps on accuracy

Figura 6. Estudio realizado de clasificador Naïve Bayes con *stopwords*.

2.4.1.3 Stemming

Es la reducción de derivaciones en palabras, quedándonos con la raíz únicamente de la misma y eliminando el sufijo o prefijo. Por ejemplo, las palabras en inglés "*connected*", "*connection*", "*connects*", … corresponderían a la palabra "*connect*".

2.4.1.4 Lemmatization

Igual que en *stemming* pero no se refiere estrictamente a sufijos/prefijos sino a palabras raíz de otras. Por ejemplo, derivaciones verbales ("*went*"; "*go*") u otras ("*best*"; "*good*").

2.4.1.5 Tokenización

La tokenización de frases consiste en dividir estas frases en cadenas más pequeñas, como por ejemplo palabras. En términos generales nos referimos a tokenización para el nivel palabra y a la segmentación cuando generamos cadenas más grandes como sentencias, párrafos, oraciones, etc.

2.4.1.6 Vectorización

Todas estas técnicas lo que pretenden es transformar en valores numéricos aquellos datos no numéricos para su posterior procesado mediante técnicas de clasificación. La utilización que hacen de distancias obliga en cierta manera a este tipo de codificación.

Ejemplos de técnicas de vectorización serían:

- *One Hot Encoding*. Es una técnica por la cual se analiza una variable categórica y se generan nuevas variables binarias que determinan si ese registro tiene o no ese valor. Imaginemos que tenemos la siguiente matriz de valores para un tipo de enfermedad:

Tipo de enfermedad	Tipo de enfermedad num
Muy leve	0
Leve	1
Medio	2
Grave	3
Muy grave	4

Generaríamos 5 nuevas columnas (hay 5 valores posibles) que indicarían si ese registro pertenece o no a la categoría (0 no pertenece /1 pertenece):

Tipo de enfermedad	Tipo de enfermedad num	Es muy leve	Es leve	Es medio	Es grave	Es muy grave
Muy leve	0	1	0	0	0	0
Leve	1	0	1	0	0	0
Medio	2	0	0	1	0	0
Grave	3	0	0	0	1	0
Muy grave	4	0	0	0	0	1

- *Bag-of-Words* (BoW) [20]. Es una técnica muy utilizada en NLP por la cual se genera una representación de valores numéricos (BoW) que indican el número de ocurrencias de palabras (o grupos de palabras) en un texto. Se denomina “bag” (bolsa) porque no hay ninguna ordenación u organización de las palabras utilizadas. Para crear un BoW será necesario:

  - Procesar el texto y extraer las palabras (o grupos) presentes en el texto.

  - Marcar cada palabra con una puntuación. Lo más simple sería marcar un 0 como no presente y un 1 como presente. Otra forma más depurada sería tener en cuenta la frecuencia de la palabra en el texto y el número de veces que aparece, y asignar un número (entre 0 y 1).

  Como el número de palabras, y dependiendo del texto a tratar, puede ser muy alto es especialmente importante realizar un buen preprocesado de los datos. Así los algoritmos trabajaran con información útil y con un rendimiento mejor en calidad y tiempos de ejecución.

  Como puntos negativos tendríamos:

  - Vocabulario. Hay que cuidar la cantidad de vocabulario a representar, ya que impacta directamente en la representación del texto a procesar.

- Dispersión. Las representaciones dispersas son más difíciles de modelar tanto por razones computacionales (complejidad de espacio y tiempo) como también por razones de información, donde el desafío es que los modelos aprovechen tan poca información en un espacio de representación tan grande.

- Significado. El significado de un texto afecta al análisis del mismo. Por esta razón el orden de las palabras puede influir en el significado del texto, así como la utilización de sinónimos.

- *Term Frequency – Inverse Document Frequency (Tf-idf)*. Es una técnica que pretende mejorar el resultado de procesos como BoW gestionando la aparición de palabras que no aportan nada al posible resultado final pero que aparecen mucho, lo cual provoca tiempos de ejecución elevados. Esta técnica asigna puntuaciones en función de su aparición en textos, de forma que las palabras que más aparecen son penalizadas en su puntuación final (por ejemplo, los artículos como "la", "el", etc.). Por el contrario, las más frecuentes son "premiadas" con mejores puntuaciones, ya que pueden dar más información del texto analizado.

- *Word2Vec*. Se trata de una técnica desarrollada en los últimos años por la cual se intenta asociar un conjunto de palabras según su similitud en el contexto, generando una dependencia de palabras con otras palabras. Se trataría de mejorar los sistemas como *One Hot Encoding* por las cuales cada palabra encontrada es independiente del resto y por tanto se considera igual de importante a la hora de clasificar un texto.
  Estos grupos de palabras generados (en inglés *embeddings*) tendrían una representación espacial similar, lo que indicaría que a más cercanía mayor similitud. Estas técnicas utilizan métodos basados en redes neuronales, en concreto *Skip Gram* y *Common Bag of Words (CBoW)*.
  En ambos casos la capa de entrada trataría las palabras únicas presentes en el texto (obtenidas mediante técnicas de preprocesamiento y tokenización), éstas serían procesadas por las capas ocultas y retornadas en la capa de salida mediante *embeddings*.
  La parte negativa es que se trata de una técnica de una exigencia alta en cuanto a recursos, ya que utiliza conceptos relacionados con redes neuronales. Se pueden encontrar numerosos tutoriales en la web, como el publicado por Zhi Li en 2019 [21].

2.4.1.7 Otras técnicas

- Contracciones. Se daría sobre todo en el caso de inglés, que por otro lado es el que cuenta con un mayor número de estudios al respecto, y se trataría de transformar expresiones del tipo "*don't*" en "*do not*" para estandarizar el texto.

- *Lowercasing*. Pasar todas las letras a minúsculas.

- Tratamiento de números. Conversión de dígitos a su representación en palabras (2 →dos).

2.4.2 Selección de características

La selección de características (en inglés *features selection*) [22] [23] es un proceso en el cual, a partir de un conjunto origen de datos, se seleccionan aquellas que nos interesen para nuestro estudio. Por tanto, se trata básicamente de eliminar características, seleccionando aquellas que nos permitan maximizar los resultados.

Hay distintos tipos de categorización para la selección de características. Entre las más destacadas encontramos:

- Según los datos de entrenamiento utilizados: supervisados, no supervisados o parcialmente etiquetados (Fig.7).

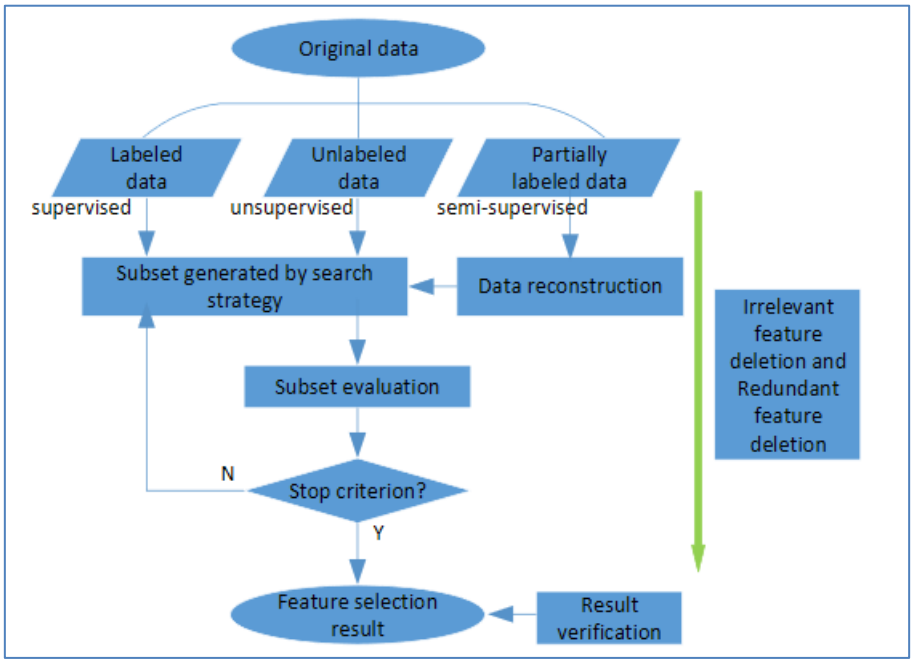


Figura 7. Tipos de selección de características.

Dentro del apartado de algoritmos supervisados podemos distinguir dos tipologías distintas:

  - Métodos univariantes. Se consideran las características de forma independiente y separada. Este tipo de métodos acostumbran a ofrecer una ordenación o ranking de las características según su bonanza en un conjunto de muestras de entrenamiento. Uno de los métodos más utilizados son los basados en test estadísticos como por ejemplo *T-statistics* o *Chi-square*.

  - Métodos multivariantes. En este caso se consideran subconjuntos de características, y se calcula la función de bonanza por cada

subconjunto. En un artículo de conferencia los autores A. Jović, K. Brkić y N. Bogunović en 2015 [24] categorizan estos métodos en algoritmos de tipo (Fig.8):

- Exponenciales. Evalúan un número de muestras que crece exponencialmente asociado a la característica a tratar.

- Secuenciales. Añaden o eliminan características de forma secuencial hasta encontrar un valor mínimo local.

- Aleatorios. Utilizan la aleatoriedad en la búsqueda, evitando el problema del valor mínimo local.

Algorithm group	Algorithm name
Exponential	Exhaustive search Branch-and-bound
Sequential	Greedy forward selection or backward elimination Best-first Linear forward selection Floating forward or backward selection Beam search (and beam stack search) Race search
Randomized	Random generation Simulated annealing Evolutionary computation algorithms (e.g. genetic, ant colony optimization) Scatter search

Figura 8. Categorización de algoritmos multivariantes

- Según su relación con modelos de ML tenemos distintos algoritmos:

  - Filtro. Las características o subconjuntos de características son evaluados de forma independiente del método de clasificación que se utiliza posteriormente.

  - Empotrados (“*wrappers*”). Se utiliza el clasificador que se usará posteriormente para evaluar qué característica o subconjunto de características es el más adecuado

  - Integrados (“*embedded*”). Los métodos integrados completan el proceso de selección de características dentro de la construcción del propio algoritmo de aprendizaje automático. En otras palabras, realizan la selección de características durante el entrenamiento del modelo.

2.4.2 Extracción de características

Continuando con la problemática por la cual tenemos información que no es útil para nuestro estudio, hemos visto en la selección de características la

posibilidad de extraer sólo aquello que nos sirva. En la extracción de características la idea es continuar en esta línea, pero en este caso tenemos la posibilidad de añadir nuevos atributos a partir de los ya conocidos. Una categorización de este tipo de técnicas es la siguiente:

- Extracción de características no supervisada. A partir de un conjunto de datos de entrenamiento el objetivo es la obtención de un espacio de dimensión reducida que minimice una determinada función de coste. Ejemplos de este tipo de técnicas son el análisis de componentes principales (PCA).

- Extracción de características supervisada. Los algoritmos disponen de información sobre la pertenencia de los datos a un determinado conjunto de clases (etiquetas asociadas a cada muestra). El objetivo es conseguir un conjunto de características que permita una separación óptima de los ejemplos de entrenamiento según este etiquetado. El ejemplo más paradigmático de esta categoría son las técnicas basadas en el análisis discriminante.

2.4.2.1 PCA. Análisis de componentes principales

El análisis de componentes principales (PCA) es posiblemente la técnica de extracción de características que más se usa. Se trata de una técnica estadística cuyo objetivo es la reducción de la dimensión o número de atributos. Se basa en el supuesto de que la mayor parte de la información de un juego de datos puede ser explicada por un número menor de variables o atributos.

Para la aplicación de PCA para un caso concreto es necesario que exista una alta correlación entre las variables, ya que esto nos habla bien a las claras que existe información redundante, por lo que con un menor número de variables podremos realizar un estudio sin perder información de los datos originales.

La selección de variables se realiza de manera que el primer factor recoja la mayor proporción posible de la variabilidad original, el segundo factor debe recoger la mayor proporción posible que no haya recogido el primero, y así sucesivamente.

Utilizaremos un ejemplo para mostrar un caso de análisis de componentes principales:

- Paso 1. Calcular la media de los atributos. De esta forma obtendremos un conjunto de datos centrado en el origen de información.
  Sea *X* la matriz de datos y *M* una matriz *MxN*, donde se ha replicado *N* veces el vector promedio n de las columnas de *X*:

$$n = \frac{1}{N}\sum_{i=1}^{N} x\_i$$

  , obtenemos:

$$\hat{X} = X - M$$

- Paso 2. Obtener la matriz de covarianza.

$$C = \frac{1}{N}\hat{X} \times \hat{X}^T$$

- Paso 3. Obtener los vectores y valores propios (*eigenvector* y *eigenvalue*) sobre la matriz de covarianza. Para no entrar en detalle estos valores nos permiten encontrar la siguiente relación:

$$\mathbf{Ax} = \lambda\mathbf{x}$$

, donde *A* sería la matriz de covarianza, *x* sería el *eigenvector* y *λ* el *eigenvalue*. En la figura 9 podemos ver una representación en eje de coordenadas de un caso concreto.

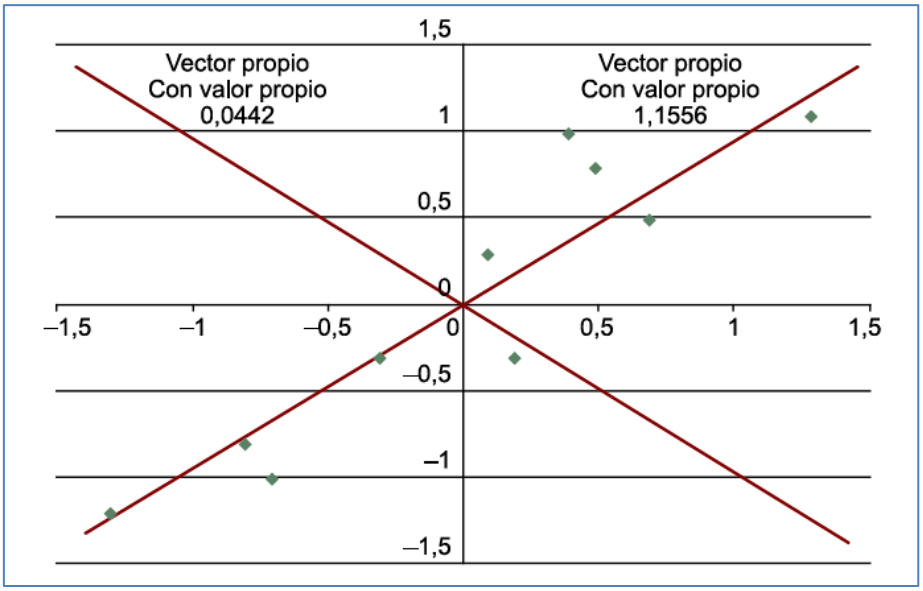


Figura 9. Representación PCA de vectores y valores propios.

- Paso 4. Proyección de los datos sobre los vectores principales. Utilizando los *eigenvectors* podremos generar una proyección utilizando los datos originales corregidos por las medias. En la figura 10 se muestra un resultado de prueba a partir de los valores PCA encontrados en la figura anterior.

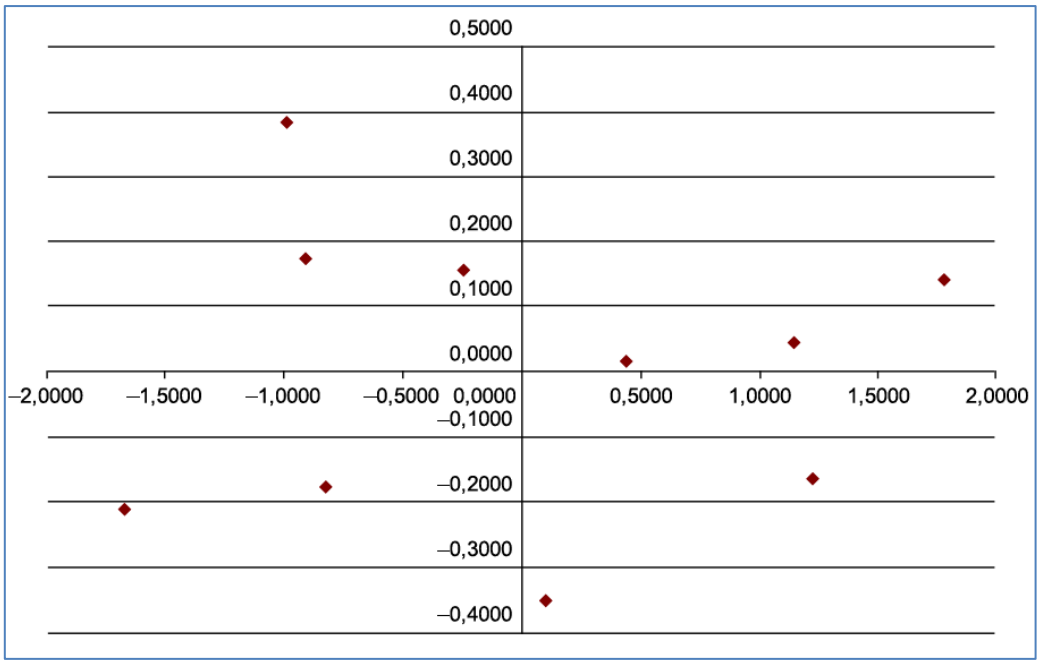


Figura 10. Proyección sobre los componentes principales.

- Paso 5. Interpretación de los resultados. Se trataría finalmente de analizar posibles patrones entre variables que nos permitan generar nuevas variables eliminando las originales.
  En los casos anteriores representados por figuras podríamos considerar que el primer vector (el considerado con valor propio 1,1556) sería el que aglutinaría la mayoría de la variabilidad total. Podemos ver que la proyección del *eigenvector* pasa muy cerca de los valores representados en el eje de coordenadas (fig.9).
  En este caso representado concreto se ha partido de un conjunto de datos de 2 variables, lo cual no sería un buen caso práctico, al quedarnos con solo 1 variable. Es en juegos de datos con muchas variables donde cobra importancia esta técnica.

2.4.3 Clasificación de sentimiento

Una vez hemos podido filtrar y depurar la información de origen llegaríamos al punto en el cual ya podemos analizar y clasificar el sentimiento. Como se ha visto hay básicamente tres enfoques que veremos a continuación.

2.4.3.1 Lexicon-based

Las técnicas basadas en léxico, aplicadas a análisis de sentimiento, se basan en la utilización de bases de datos de sentimientos (en inglés *sentiment lexicons*) para obtener la polaridad de las opiniones en textos. Como hemos visto anteriormente una de las técnicas a utilizar es la tokenización, por la cual seleccionamos palabras o grupos de palabras dentro de un texto. Estos *tokens* serían los buscados en las bases de datos, obteniendo su polaridad. Posteriormente debería ser el proceso de análisis quien fuera asignando pesos a los distintos *tokens* para finalmente obtener la polaridad del texto.

Como bases de datos de léxicos más utilizadas tendríamos *SentiWordNet*, *WordNet-Affect*, *MPQA* y *SenticNet*, analizadas en el trabajo de Cataldo Musto, Giovanni Semeraro y Marco Polignano en 2014 [25].
Los tipos de *lexicon* utilizados serían:

- Basados en diccionario. La estrategia seria la siguiente:

    - Se selecciona un grupo de palabras de las cuales se conoce su polaridad.

    - Se buscan esas palabras en las bases de datos comentadas anteriormente, obteniendo sus sinónimos y antónimos.

    - Estos sinónimos y antónimos se añaden a la lista y se vuelve a realizar otra iteración.

    - Cuando no haya nuevas palabras el proceso finaliza.

  La utilización de estos diccionarios tiene un problema y es que no es capaz de encontrar la opinión correcta de una palabra por el dominio y el contexto en que se encuentra. Por ejemplo, la utilización de los adjetivos grande y pequeño puede considerarse como positivo o negativo según el concepto sobre el que se quiere opinar. A este respecto una de las posibles soluciones sería la de establecer estrategias basadas en diccionarios genéricos, pero aplicando las necesarias correcciones dentro del ámbito de aplicación, lo que conlleva a la generación de nuevos diccionarios focalizados.

- Basados en *Corpus*. Este enfoque ayuda a resolver el problema de encontrar palabras de opinión con orientaciones específicas del contexto. Sus métodos dependen de patrones sintácticos o patrones que ocurren junto con una lista de semillas de palabras de opinión para encontrar otras palabras de opinión en un corpus mayor.
  Uno de estos métodos fue representado por V.Hatzivassiloglou y K.McKeown en 1997 [26] , por el cual, a partir de una lista de adjetivos de opinión semilla en inglés aplicaron un conjunto de restricciones lingüísticas con el fin de identificar palabras de opiniones adjetivas adicionales y sus orientaciones. Las restricciones utilizadas eran para conectivos del tipo “y”, “o”, “pero”, etc. Uno de los hallazgos encontrados es que la conjunción “y” (“and” en inglés) dice que los adjetivos unidos suelen tener la misma orientación. Esta idea se llama coherencia de sentimiento, aunque no siempre es coherente en la práctica. También hay expresiones adversas como “pero” o “sin embargo”, que se indican como cambios de opinión. Para determinar si dos adjetivos unidos son de la misma o diferente orientación, el aprendizaje se aplica a un gran corpus. Luego, los vínculos entre los adjetivos forman un gráfico y el agrupamiento se realiza en el gráfico para producir dos conjuntos de palabras: positivo y negativo.

Sin embargo, esta técnica basada en *Corpus* tiene una desventaja respecto a la fundamentada en diccionario, y es que es muy difícil construir un gran corpus que cubra todo el espectro de un idioma.

2.4.3.2 ML no supervisado

La clasificación no supervisada [27] persigue la obtención de un modelo válido para clasificar objetos a partir de la similitud de sus características. A partir de un conjunto de objetos descritos por un vector de características y a partir de una métrica que nos defina el concepto de similitud entre objetos, se construye un modelo o regla general que nos va a permitir clasificar todos los objetos. En este caso no se trata de predecir sino de descubrir patrones.

2.4.3.2.1 Dendrogramas

Un dendrograma es un diagrama que muestra las agrupaciones sucesivas que genera un algoritmo jerárquico aglomerativo. En ellas se establece una relación entre las categorías, que se expresa mediante una estructura de árbol. Este tipo de algoritmo nos define conceptos a distintos niveles, y nos establece que los conceptos que comparten un nodo padre son más semejantes entre ellos que con los demás. Además, se entiende que la relación entre un nodo y su nodo padre en una jerarquía corresponde a una relación de superclase. En la figura 11 se puede observar una representación para un conjunto de datos *X={x1,...,x5}.*

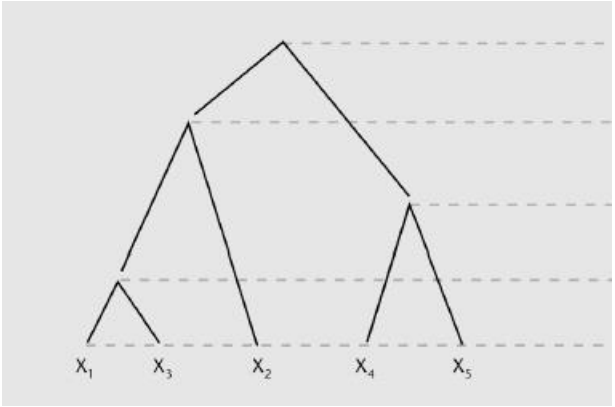


Figura 11. Representación gráfica dendrograma

2.4.3.2.2 K-Means

Para empezar, es necesario definir el número de grupos (*K*) que se quieren obtener. Supongamos que disponemos de un juego de datos compuesto por *n* casos, donde cada caso está formado por una serie de atributos que lo caracterizan (por ejemplo, información personal de personas: altura, edad, etc.). Si consideramos *X={x1,…,xn)*, donde *n* serian el número de casos (personas) y cada caso tiene m atributos. Para clasificar nuestro juego de datos *X* mediante el algoritmo k-means seguiremos los siguientes pasos:

- De entre los n casos seleccionaremos k, que llamaremos semillas. Cada semilla identificará su grupo o *cluster*.

- Asignaremos el caso al clúster cuando la distancia entre el caso y la semilla sea la menor de entre todas las semillas.

- Calcular la mejora que se produciría si asignáramos un caso a un clúster al que no pertenece actualmente. Hay diferentes criterios para evaluar esta mejora, uno de ellos podría ser el de minimizar la distancia de las distintas instancias o casos a sus respectivos centros.

- Hacer el cambio con la mejora.

- Repetir los dos pasos anteriores hasta que no haya cambios que provoquen mejora.

El algoritmo puede provocar diferentes soluciones según la métrica utilizada para la distancia y el criterio de mejora aplicado. De la misma forma si el número de grupos (*K*) no está bien definido o el conjunto de datos no tiene ninguna similitud para poder agrupar en clústeres, el resultado no será óptimo.

2.4.3.3 ML supervisado

Esta tipología de algoritmos [28] se basa en el hecho por el cual disponemos de un conjunto de ejemplos y para cada ejemplo conocemos el valor de salida. Por tanto, la clasificación supervisada persigue la obtención de un modelo válido para predecir casos futuros a partir del aprendizaje de casos conocidos.

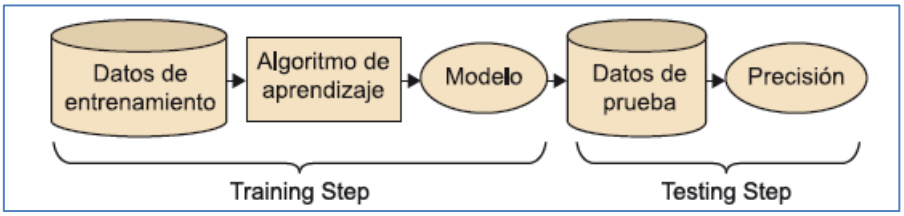


Figura 12. Aprendizaje supervisado.

En la fig.12 se puede apreciar que en una secuencia de aplicación de un modelo de aprendizaje supervisado es a partir de un juego de datos de entrenamiento donde se ajusta el modelo de aprendizaje, que acaba estableciendo un modelo. Finalmente, el nivel de exactitud es evaluado a partir de un juego de datos de prueba, distinto al de aprendizaje. El juego de datos típico de una instancia está formado por atributos descriptivos y un atributo objetivo también llamado clase. Para los casos de análisis de sentimiento se suele utilizar la nomenclatura de dominio {0,1} para denotar la polaridad positiva o negativa.

Son este tipo de algoritmos los que se utilizan básicamente para problemas de análisis de sentimiento. A continuación, mostramos los más destacados.

2.4.3.3.1 K-Nearest Neighbor (K-NN)

En contraste con otros algoritmos de aprendizaje supervisado, K-NN no genera un modelo fruto del aprendizaje con datos de entrenamiento, sino que el aprendizaje sucede en el mismo momento en el que se prueban los datos de test. A este tipo de algoritmos se les conoce como métodos de aprendizaje perezosos (del inglés *lazy learning methods*). Su propia tipología denota su mayor debilidad, y es la lentitud en el proceso de clasificación.
Su funcionamiento sería como sigue:

- Para un conjunto *X* de datos de entrenamiento, sobre el que no vamos a realizar ningún proceso en específico, seleccionamos una muestra *x*.
- Se computan las distancias entre esta muestra y el resto. Se seleccionan aquellas que están a menor distancia.
- La muestra *x* se asigna a la clase más frecuente de entre aquellas instancias seleccionadas como más cercana.

Se trataría de un algoritmo iterativo en el que se irían refinando los resultados a partir de generación de agrupaciones hasta llegar a unas agrupaciones estables.

En este caso también es muy importante la selección de una *K* correcta, ya que sino los resultados pueden tener un significado erróneo. En la figura 13 podemos observar un ejemplo en la cual se trataría de agrupar la bola con signo "+" entre el resto de las bolas de color blanco y rojo. En el caso de k=1, la bola estaría agrupada como blanca, ya que estaría en la zona en la que sólo habría casos de bolas blancas. En el caso de k=2 la zona se amplía y pasamos a tener casos de bolas rojas, por lo que no podemos decir a qué grupo se clasificaría. Para el caso de k=3 al haber más bolas rojas en la zona se clasificaría la bola como roja.

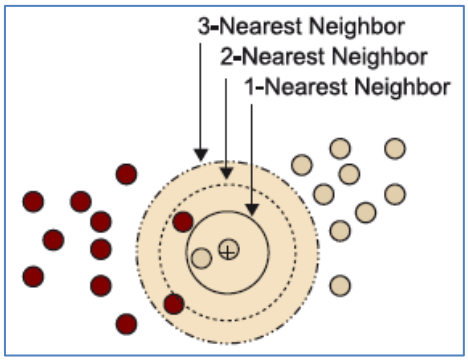


Figura 13. K-NN ejemplo.

2.4.3.3.2 Árboles de decisión

Desde un punto de vista visual estos tipos de algoritmos tienen una representación gráfica muy clara que ayuda a entender los resultados.
Un árbol de decisión se fundamenta en un nodo principal construido a partir de uno de los atributos del juego de datos, como el que se representa en la figura 14 de ejemplo con el campo "¿Edad?", y sus posibles valores ("joven", "medio"

y “mayor”) que se relacionarían con más nodos. A partir de aquí se van desarrollando los nodos u hojas del árbol hasta llegar al final de la clasificación de atributos. Es en este momento cuando podemos evaluar el resultado de nuestras muestras. Por ejemplo, tendríamos 2 casos de persona joven y con trabajo que se han clasificado bien dentro del árbol.

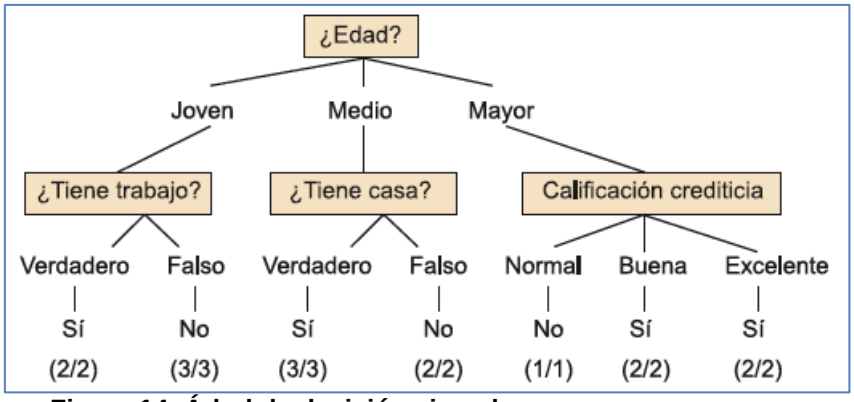


Figura 14. Árbol de decisión ejemplo.

Para el caso que nos ocupa hemos generado un árbol de ejemplo, pero podría ser que fueran aplicables otros árboles distintos que también explicaran nuestro juego de pruebas.

Uno de los puntos débiles de este tipo de algoritmos es la posibilidad de sobre entrenamiento del modelo. El sobre entrenamiento se da cuando el modelo está demasiado basado en los datos de prueba, lo que provoca que el modelo sea bueno con estos datos, pero malo a la hora de tratar otros datos (p.e. los datos de prueba o test). Una de las posibles soluciones a esto y que también es muy utilizado para optimizar la velocidad de un algoritmo basado en árboles de decisión es el concepto de poda. Este concepto se basa en la idea de medir el error estimado de cada nodo, de modo que, si el error estimado para un nodo es menor que el error estimado para sus subnodos, entonces los subnodos se eliminan.

2.4.3.3.3 Redes neuronales

Las redes neuronales [29] no son más que un intento de trasladar el comportamiento de la sinapsis cerebral humana al terreno del *Machine Learning*. Son un conjunto de algoritmos diseñados para reconocer patrones que reconocen la información de entrada que atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida. Todas estas neuronas están conectadas entre ellas a través de enlaces donde el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. A su vez pueden estar agrupadas en distintas capas (lo que se conoce como *Deep Learning*), cuyas tareas se combinarían para retornar un resultado (observar la figura 15 ante un problema de reconocimiento facial).
Su principal ventaja es que permiten aproximar funciones de respuesta ante problemas de clasificación o *clustering* sin un modelo pre-generado. De hecho,

actúa de “aproximador” a funciones que pueda resolver problemas del tipo *f(x)=y* donde la función en cuestión no se conoce a priori. Su utilización es diversa, desde el reconocimiento facial o de voz al procesamiento del lenguaje natural. Otra ventaja es su escalabilidad, debido en parte a su arquitectura en capas.

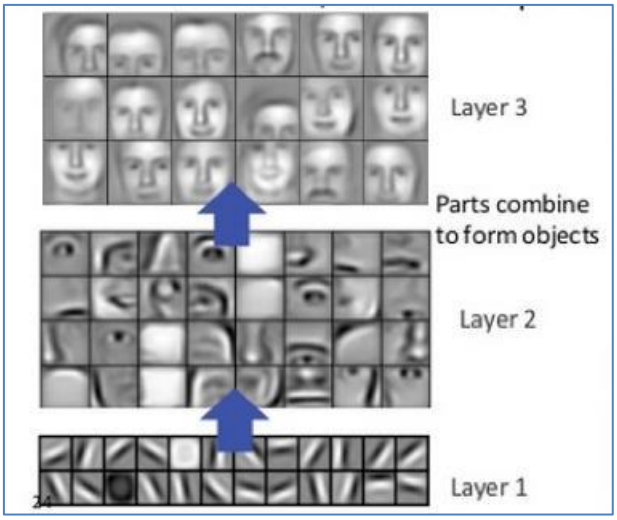


Figura 15. Ejemplo de capas relacionadas con Deep Learning.

Por otro lado, en cuanto a desventajas podemos considerar que son algoritmos que necesitan de grandes datos para el entrenamiento y, asociado a esto, también es necesaria una gran potencia de cálculo, reservada en muchas ocasiones a máquinas de altas capacidades.

2.4.3.3.4 Support Vector Machines (SVM)

SVM es una de las técnicas más conocidas de algoritmos predictivos. Son usadas normalmente como clasificadores y se fundamentan en la búsqueda de un plano (o hiperplano) que “divida” los datos en dos, permitiendo segregar los datos en grupos. Son usadas en aplicaciones tales como reconocimiento facial, clasificación de emails o de genes, por poner algunos ejemplos. Son capaces de producir buenos modelos para resolver problemas de clasificación binaria, pero también para tareas de regresión, de multiclasificación y de agrupamiento. Con todo ello las SVM son consideradas como los mejores algoritmos para aplicaciones de clasificación de texto. Sin embargo, solo funcionan para espacios numéricos, de forma que para atributos categóricos será necesario un proceso previo de conversión de valores categóricos a numéricos.

Las SVM son más rápidas en comparación con otras técnicas como *Naïve Bayes* o *Logistic Regression*, y por otro lado, son más óptimos en cuanto a uso de memoria al utilizar solo un subconjunto de datos para el entrenamiento.

Como puntos negativos se considera que consumen un elevado tiempo en la construcción de modelos lo cual no la hace elegible para conjuntos de datos masivos. Por otro lado, está muy condicionada al tipo de *kernel* (tipo de función que se utiliza para separar los conjuntos de datos) utilizado, y no es muy útil cuando el conjunto de datos tiene clases poco definidas.

En cuanto a su nivel de interpretación es bastante complejo, ya que, dependiendo del caso, los hiperplanos producidos son de múltiples dimensiones.

2.4.3.3.5 Naïve Bayes

Se trata de un clasificador que simplifica enormemente el aprendizaje al asumir que las características son independientes de la clase dada. Esta asunción, arriesgada en principio, se ha demostrado en la práctica como muy acertada, como se demuestra en el trabajo de I.Rish en 2001 [30]. *Naïve Bayes* arroja resultados mejores que técnicas mucho más sofisticadas, y para el caso que nos ocupa en este trabajo, resultados muy buenos para clasificación de textos.

El modelo bayesiano de probabilidad condicionada se suele basar en este concepto:

$$P(A|B)=P(A\cap B)/P(B)$$

, donde, la probabilidad de que se dé el caso A dado B es igual a la probabilidad de la intersección de A con B (A∩B) partido la probabilidad de B.

Desarrollando esta base tendríamos la fórmula utilizada en el teorema de Bayes, explicada en detalle en el artículo web de J.A.Camacho en 2020 [31] :

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

Figura 16. Teorema de Bayes

, donde:

- P(H) es la probabilidad a priori, la forma de introducir conocimiento previo sobre los valores que puede tomar la hipótesis.
- P(D|H) es la probabilidad de obtener D dado que H es verdadera.
- P(D) es la probabilidad de observar los datos D promediado sobre todas las posibles hipótesis H.
- P(H|D) es la probabilidad a posteriori, o sea, la distribución de probabilidad final para la hipótesis.

Este método es muy simple y uno de los más utilizados para tareas de clasificación, pero también tiene algunos inconvenientes como el hecho de que asume que todos los textos son subjetivos y que las características son independientes entre sí (ya comentado).

Por otro lado, es un método que funciona bien con datos continuos, es decir, aquellos que ocupan un espacio en un continuo, pero no tienen valores fijos como puede ser por ejemplo la altura de una persona. En cambio, si se ha de trabajar con datos discretos (valores fijos) es preferible utilizar una variante que es la conocida como *Multinomial Naïve Bayes*. Con esta variante se considera el número de apariciones del término para evaluar la contribución de la probabilidad condicional dada la clase, por lo que el modelado se ajusta mejor a la clase.

2.4.3.3.6 Logistic Regression

Logistic regression [32] es un clasificador que mejora las capacidades de *Linear Regression*, y que se aplica a búsqueda de valores binarios (0,1). En este sentido, el problema con *Linear Regression* es que trabaja bien con resultados 0 ó 1, es decir, es capaz de encontrar un hiperplano que divida los grupos de muestras encontrando las distancias mínimas. Pero esta funcionalidad no es demasiado buena en el caso que queramos usarla como clasificador, ya que no es capaz de aplicar probabilidades. Por tanto, no es correcta utilizarla en problemas de clasificación con múltiples clases.

Para solucionar esto aparece *Logistic Regression*. Mientras que *Linear Regression* genera un hiperplano recto, *Logistic Regression* utiliza la función logística (Fig.17) para representar adecuadamente los valores que se pueden encontrar entre 0 y 1:

$$\text{logistic}(\eta) = \frac{1}{1 + exp(-\eta)}$$

Figura 17. Función logística.

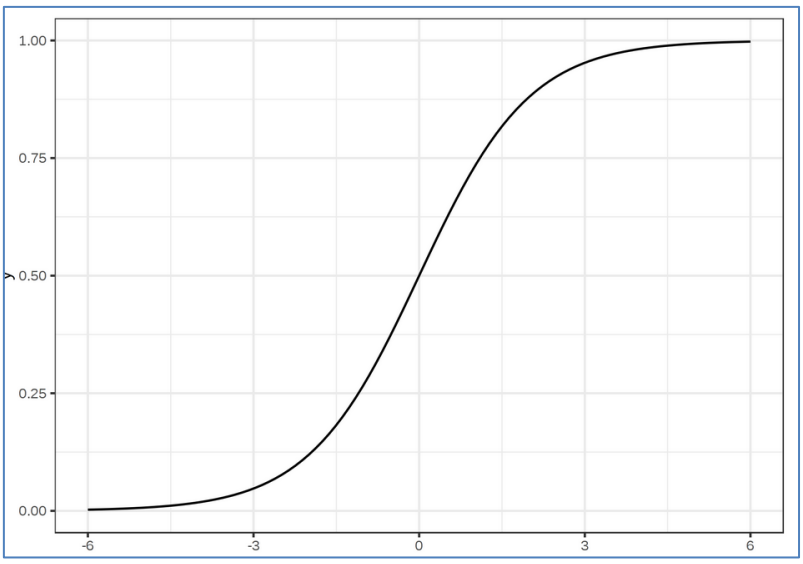


Figura 18. Representación de la función logística.

Logistic Regression tiene algunas desventajas como es el llamado “separación completa” (del inglés *complete separation*), por el cual si hay algún campo que divide exactamente dos clases, el modelo no lo representará correctamente, ya que el peso para esa característica no convergería al tener un peso óptimo con tendencia al infinito. Otra desventaja es la existencia probada de otros modelos que obtienen mejores resultados.

Sin embargo como se puede comprobar en diferentes *tests*, como el realizado por J.Lin y A.Kolcz en 2012 [33] para un conjunto de datos formado por “tweets”, los resultados pueden ser muy buenos en análisis relacionados con textos.

Como punto fuerte hemos de considerar el hecho que *Logistic Regression* no es solo un clasificador simple, sino que también retorna probabilidades, lo que es muy útil para valorar si una instancia concreta tiene mayor o menor probabilidad de pertenecer a una clase.

Como hemos comentado *Logistic Regression* es un clasificador binario que aumenta las capacidades de *Linear Regression*. En el caso de necesitar un clasificador multi-clase utilizaríamos *Multinomial Regression*.

2.4.3.3.7 ULMFiT

Como añadido enumeramos a continuación una de las técnicas aparecidas más recientemente y que se engloban dentro del llamado *Transfer Learning*. En el terreno de NLP un paso más respecto a las técnicas tradicionales de ML es el uso de *Deep Learning* o algoritmos basados en redes neuronales. Por encima incluso, y de hecho aplicando las bases de *Deep Learning*, han aparecido diferentes técnicas de *Transfer Learning* basadas en la generación de modelos de lenguaje pre-entrenados. La diferencia con técnicas clásicas es el reaprovechamiento de conocimiento adquirido previamente para “enriquecer” los modelos.

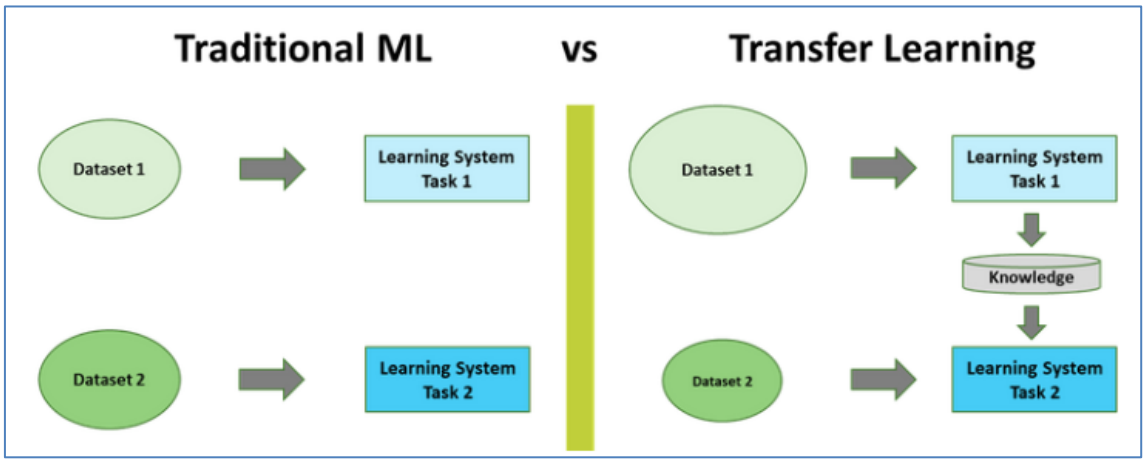


Figura 19. ML tradicional vs Transfer Learning.

ULMFit [34] [35] concretamente, realiza el proceso de aprendizaje en 3 etapas:

- *Language Model Pre-Training*. El modelo se pre-entrena con un *dataset* muy grande, pero de tipo genérico. El modelo entonces es capaz de predecir, con un cierto grado de acierto, la siguiente palabra en una secuencia. En este caso podríamos considerar que el modelo conoce un cierto patrón en la construcción de frases, pero a nivel genérico.

- *Language Model Fine-Tuning*. En este paso, y reaprovechando el conocimiento del anterior paso, se generaría un modelo más enfocado al texto de estudio. Por ejemplo, sería cuando generáramos el modelo relativo a textos en Twitter, los cuales tienen unas reglas sintácticas propias (existencia de jerga, *links*, direcciones de Twitter, etc.). De la misma forma que el paso anterior tendríamos un modelo capaz de predecir la siguiente palabra en una secuencia.

- *Text classifier Fine-tuning*. Se realiza la tarea de clasificar el texto según el objetivo deseado aprovechando el conocimiento de los dos pasos anteriores. Aplicándolo a nuestro trabajo concreto, sería el turno de realizar la clasificación de sentimiento: positivo, negativo o neutro.

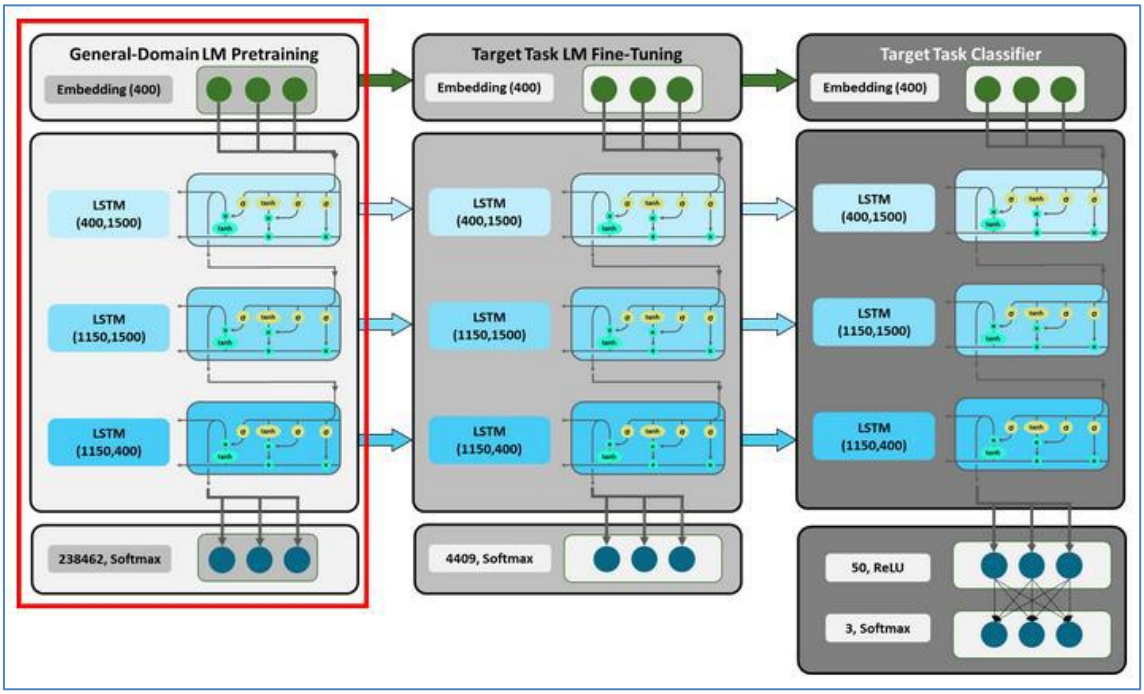


Figura 20. Ejemplo de arquitectura ULMFiT.

Uno de los problemas al hacer *Deep Learning* y *Transfer Learning* es la cantidad de información a procesar. Hay que considerar que en *Transfer Learning* toda la información aprendida tiene que ser traspasada entre capas por lo que el tiempo de proceso es muy alto.
Dado que es necesario procesar mucha información para obtener resultados correctos, en ULMFiT se recomienda realizar algunas técnicas para el *Fine-tuning*. Entre ellas destaca lo que se denomina *freezing*, que como su propio nombre indica en inglés, se trata de desactivar (*freeze*) y activar capas (*layers*)

de nuestro algoritmo de más a menos genérica, activándolas a medida que el resultado del modelo se ajusta y converge. Con esto se consigue no sobreentrenar el modelo, yendo paso a paso en la asignación de pesos en los componentes de la red neuronal.

Algunas ventajas de este tipo de técnicas serian:

- Hay una compartición de conocimiento, por lo que se puede reaprovechar, mientras que el ML “tradicional” está muy enfocado al *dataset* de estudio.

- Menos datos de entrenamiento. Dado que ya tenemos modelos genéricos, podemos aprovechar este conocimiento para, a partir de aquí, aplicarlo a nuestro caso concreto. No sería por tanto necesario el proceso de tanta información. Imaginemos que queremos desarrollar un algoritmo para el reconocimiento de texto en Twitter en portugués. Dado que seguramente existirán algoritmos genéricos de reconocimiento de textos en portugués podremos aplicarlo para enriquecer nuestro modelo aplicado al caso concreto de Twitter.

- Menos tiempo de proceso. Dado que no se parte de cero a la hora de construir un modelo, sino que se reaprovecha el conocimiento de alguno existente, inicialmente el tiempo de proceso debería ser inferior.

- Robustez. Al partir de un modelo base que ya funciona, y considerando que éstos están bien ajustados, nuestro resultado final enfocado a nuestra problemática debería tener más posibilidades de éxito.

- Los modelos generalizan mejor. La generalización es la capacidad de analizar textos no entrenados previamente con una buena tasa de acierto. Dado que parten de unos modelos generales (y, en teoría, bien ajustados) son capaces de detectar patrones que pueden aplicar en contextos nuevos.

2.4.4 Evaluación de resultados

Para las técnicas de aprendizaje supervisado, y especialmente en problemas de clasificación, es importante establecer un protocolo para validar hasta qué punto el sistema ha aprendido bien un conjunto de datos.

Podríamos decir que hay dos tipos de validación actualmente:

- Validación simple. Se trataría de dividir los datos de entrada en dos: datos de entrenamiento y de *test*. Los primeros son los utilizados para generar el modelo. Una vez obtenido se comprueba contra los datos de *test*. Estas comprobaciones se basan en las clasificaciones que ha hecho el modelo de los datos de *test* y que se cuantifican mediante las diferentes medidas de evaluación que veremos a continuación.

Normalmente una proporción 80-20 (80% datos de entrenamiento y 20% en datos de *test*) sería una división suficiente para estas validaciones. La parte negativa es que puede haber diferencias en caso de utilizar unos datos u otros en la división entre entrenamiento y test. Por tanto, es recomendable si no se conocen los datos, y al no poder revisarlos todos caso por caso, utilizar herramientas de selección de muestras aleatorias, pero siempre utilizando la misma selección.

- Validación cruzada. Surge con el afán de minimizar el hecho de la variabilidad en los resultados según la división del conjunto de datos seleccionado.
  Para ello divide el conjunto de datos en k subconjuntos. A continuación, se hacen k pruebas utilizando en cada una un subconjunto como test y el resto como entrenamiento. A partir de ahí, se calcula la media y la desviación estándar de los resultados. Las k seleccionadas varían según la cantidad de información que se tiene, pero un valor alrededor de 10 suele ser suficiente. Hay que tener en cuenta que cuanto más grande sea el valor k (más subconjuntos), el proceso de validación tardará más.

La mayoría de las medidas de evaluación se pueden expresar en función de la matriz de confusión (fig.21). La matriz de confusión contiene una partición de los ejemplos en función de su clase y predicción.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- **VP** es la cantidad de *positivos* que fueron *clasificados correctamente* como positivos por el modelo.
- **VN** es la cantidad de *negativos* que fueron *clasificados correctamente* como negativos por el modelo.
- **FN** es la cantidad de *positivos* que fueron *clasificados incorrectamente* como negativos.
- **FP** es la cantidad de *negativos* que fueron *clasificados incorrectamente* como positivos.

Figura 21. Matriz de confusión.

A partir de la matriz de confusión podemos encontrar distintas medidas que nos ayudarán a evaluar la bondad de nuestro modelo automatizado.

- Error. Mide el número de ejemplos mal clasificados del total. Se pretende que los algoritmos de aprendizaje tiendan a minimizar esta medida. La fórmula es:

$$\text{error} = \frac{fp + fn}{tp + fp + tn + fn}$$

- Exactitud (*Accuracy*). Mide el número de ejemplos bien clasificados del total.

$$\text{exactitud} = \frac{tp + tn}{tp + fp + tn + fn}$$

- Precisión. Mide el número de ejemplos bien clasificados del total de ejemplos con predicción positiva.

$$\text{precisión} = \frac{tp}{tp + fp}$$

- Sensibilidad (*Recall*). Mide el número de ejemplos bien clasificados del total de ejemplos positivos.

$$\text{sensibilidad} = \frac{tp}{tp + fn}$$

- F1. Basada en la precisión y la sensibilidad, esta medida descarta los elementos negativos debido al sesgo producido por la diferencia entre el número de ejemplos negativos y positivos.

$$F1 = 2 \times \frac{\text{precisió} \times \text{sensibilidad}}{\text{precisión} + \text{sensibilidad}} = \frac{2 \times tp}{2 \times tp + fn + fp}$$

2.4.5 Conclusiones del análisis

A partir de lo visto en apartado consideramos los siguientes puntos para el inicio del desarrollo:

- Como primer paso seria necesario un preprocesamiento de los datos, esto nos permitirá eliminar ruido y de pasada optimizar los procesos en tiempo. Considerando que nuestro texto procede de lenguaje cuotidiano y que se situa en un foro de opinión particular, es practicamente seguro que hay información no utilizable (*links*, caracteres especiales, etc.).

- Stemming/Lemmatization. No hay un consenso de qué técnica es más adecuada. De hecho en nuestra propia experiencia no hemos podido conseguir unificar un criterio. Después de revisar diferentes estudios conseguimos encontrar este de Kantrowitz,M., Mohit, B. y Mittal, V. en el año 2000 [36] en el que se utilizan diferentes pruebas en combinación con vectorización Tf-idf y *stemming*. La prueba consiste en generar

diferentes clases de *stems* relacionados con palabras. Se evalua el *stem* y si este deriva a la primera palabra de su clase se le da por correctamente asignado. Los resultados finales son bastante buenos, entre el 78% y el 93% de efectividad, por lo que lo consideraremos en el estudio.

- ML Supervisado. En este caso el proyecto ya estaba enfocado para la utilización de estos tipos de técnicas. En el apartado 3.3.1 se darán más detalles de la selección final realizada. En este sentido sí que hemos de notar la existencia de una serie de técnicas de ML basadas en redes neuronales, que pueden ser interesantes a la hora de comparar rendimiento con las técnicas más “clásicas”.

- Validación simple o cruzada. Uno de los puntos a favor de la validación cruzada, por su tipología, es que al utilizar diferentes capas (o conjuntos de datos) es menos probable que el modelo resultado se sobreajuste (del inglés *overfit*) a los datos. Sin embargo, este tipo de validaciones son más costosas en tiempo de máquina ya que tiene que generar n entrenamientos, mientras que la simple no.

3. Desarrollo *Back-end*

Para el desarrollo se utilizará Python3 como lenguaje de programación, dada su extendida utilización para temas relacionados con el ML. Su gran aceptación en buena medida se debe a la amplia comunidad de desarrolladores que por detrás se encarga de generar librerías de fácil uso para los analistas.
Para la parte de desarrollo y pruebas no exigentes se ha utilizado un equipo portátil PC con las siguientes características:

- CPU: Intel Core i7-6600U 2 cores, 4 threads, Base clock 2.8 GHz, turbo 3.2 GHz
- GPU: Intel HD 520 (Mobile Skylake) 1GB
- RAM: 16GB 2133 MHz
- Drive: SSD 1Tb
- SO: Windows 10

Para pruebas exigentes de modelado utilizando técnicas complejas se ha utilizado la solución en nube "*Google Colab*", utilizando procesadores de alto rendimiento de tipo GPU y una RAM de 12Mb.

Para obtener información sobre el código fuente generado consultar el punto "8.Anexos" de este documento.

3.1 Definición de datos

Nuestro *dataset* seleccionado para el estudio se encuentra en un repositorio web de Kaggle[3] específico para el análisis de sentimiento de opiniones de películas. Kaggle es uno de los repositorios más utilizados en la comunidad de análisis de datos, y cuenta con un amplio catálogo de información.
La información se basa en opiniones realizadas en el portal de internet IMDB (*Internet Movie Database*), utilizada en exclusiva como base de datos de información relacionada a la filmografía.

El *dataset* se compone de tres ficheros con extensión "csv":

- "Train" → Entrenamiento
- "Test" → Prueba
- "Valid" → Validación

, inicialmente se utilizarán los dos primeros para generar nuestros modelos. Por tanto, y a partir de ahora, el estudio se basará en ellos, a menos que se especifique lo contrario.

Estos ficheros se componen de 2 campos separados por el carácter ",":

[3] https://www.kaggle.com/columbine/imdb-dataset-sentiment-analysis-in-csv-format

- "*Text*" → Comentario sobre la película.
- "*Label*" → Etiquetado. Indicador de sentimiento (0-negativo, 1-positivo).

3.1.1 Captación inicial de datos

Utilizamos la función definida "*read_data()*" que nos permitirá leer los ficheros comentados, incluso nos permitirá filtrar registros para optimizar tiempos en las pruebas. La función "*read_csv()*" de la librería "pandas" nos permite leer los ficheros csv disponibles, sin ser necesario definir parámetros adicionales en la llamada, ya que los que vienen por defecto son suficientes (primera línea es la cabecera y el separador de campos es ",").
Se opta por retornar los campos de texto y etiquetado en dos *dataframes* separados para facilitar el estudio.

3.1.2 Descripción de datos

Utilizaremos las funciones de la librería "pandas", "*head()"* y "*describe()"*, para analizar la información de los ficheros ingestados.

Para el fichero de entrenamiento tenemos 40.000 registros de los cuales 39.723 diferentes. Al ser pocos registros no conseguiríamos demasiada mejora en tiempo de proceso, por lo que optaremos por no filtrar los datos repetidos.

```
########## Resume   #################
9311     I do not know what some of these filmmakers ar...
10587    Scooby Doo is undoubtedly one of the most simp...
21276    I happen to have read all of Junji Ito's Engli...
26580    Spheeris used this documentary to push a stere...
19566    Red Eye is a good little thriller to watch on ...
Name: text, dtype: object
########## Describe #################
count                                                40000
unique                                               39723
top       Loved today's show!!! It was a variety and not...
freq                                                     4
Name: text, dtype: object
########## Resume   #################
17131    1
2517     1
12131    1
15181    0
19230    1
Name: label, dtype: int64
########## Describe #################
count    40000.000000
mean         0.499525
std          0.500006
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: label, dtype: float64
```

Para el fichero de *test* tenemos 5.000 registros de los cuales 4.995 diferentes.

```
########## Resume   #################
1134    I hadn't heard anything about this project unt...
4386    Forgive me for stating the obvious, but some f...
2554    Is this a game FMV or a movie? In all honesty,...
3910    Loosely based on the James J Corbett biography...
3939    Aimless teens on summer break in a small Ohio ...
Name: text, dtype: object
########## Describe #################
count                                                5000
unique                                               4995
top        .......Playing Kaddiddlehopper, Col San Fernan...
freq                                                    2
Name: text, dtype: object
########## Resume   #################
1134    0
4386    0
2554    0
3910    1
3939    0
Name: label, dtype: int64
########## Describe #################
count    5000.000000
mean        0.501000
std         0.500049
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: label, dtype: float64
```

3.1.3 Verificación calidad de datos

Continuando con la función "*lookup_data()*" revisamos si hay nulos con la función *"isnull()"*:

```
########## Nulls    #################
0
```

Confirmamos que no hay en ninguno de los dos ficheros, por tanto, no realizaremos actuaciones.

3.1.4 Análisis exploratorio de datos

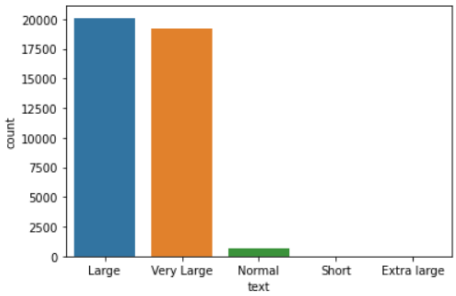
A continuación, analizaremos la distribución de los datos, y en el caso de los textos sus longitudes.

Para el caso de los datos de texto vemos que el tamaño es bastante grande, de hecho, algunos excesivamente grandes (¡más de 12.000 caracteres!), lo que nos hace plantearnos hacer un filtrado. Hemos hecho el siguiente agrupado de tamaños:

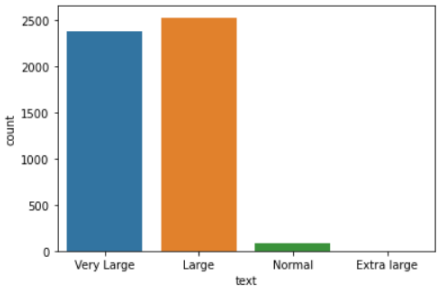
- Inferior a 4 → Muy corto
- Entre 4 y 50 → Corto
- Entre 51 y 250 → Normal
- Entre 251 y 1.000 → Largo
- Entre 1.001 y 10.000 → Muy largo
- Superior a 10.000 → Extralargo

, y obtenemos:

```
########## Max Length   ##################
13704
########## Plot ##################
Large          20117
Very Large     19250
Normal           625
Extra large        5
Short              3
Name: text, dtype: int64
```

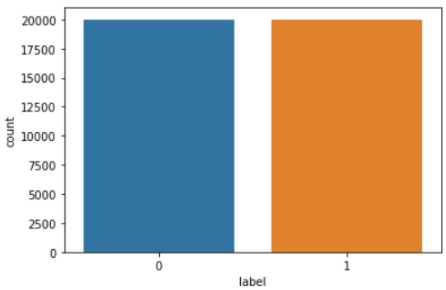



```
########## Max Length   ##################
12930
########## Plot ##################
Large          2533
Very Large     2383
Normal           83
Extra large       1
Name: text, dtype: int64
```

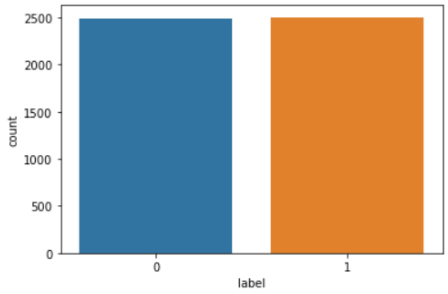


Para el caso de los etiquetados vemos que los valores están bastante balanceados (cerca del 50% de distribución entre comentarios etiquetados positivos y negativos).

```
##########Plot ################
0    20019
1    19981
Name: label, dtype: int64
```



```
###########Plot ################
1    2505
0    2495
Name: label, dtype: int64
```



Inicialmente no filtraremos información por tamaño, ya que, como se ha podido comprobar el tamaño medio del texto es bastante grande de por sí. De ser así deberíamos filtrar un gran número de registros y seguramente perderíamos fiabilidad en el sistema.

3.2 Preparación de datos

En este punto procederemos a realizar los primeros tratamientos sobre los *dataset* seleccionados. Como resultado obtendremos los datos listos para ser procesados con las técnicas de ML seleccionadas. En este punto destaca la parte de preprocesado de datos que explicamos a continuación en el apartado 3.2.2, ya que dependiendo de las técnicas aplicadas puede influir en los resultados finales.

3.2.1 Selección de datos

Como ya hemos comentado se ha habilitado una constante ("*ROWS_TO_PROCESS*") en el aplicativo, que nos permitirá procesar menos registros, y de esta manera reducir tiempos en las pruebas iniciales.

3.2.2 Limpieza de datos

Aplicaremos una serie de técnicas, ya vistas en el apartado 2.4.1, por la cual eliminaremos información no necesaria o, cuanto menos, no beneficiosa para nuestros procesos de modelado.

A nivel de caracteres se ha eliminado lo siguiente:

- *Links* html.
- Marcas de *html* ("*tags*").
- Expresiones entre corchetes.
- Caracteres no ASCII.
- Eliminación de guiones ("-") entre caracteres.
- Espacios en blanco.
- Caracteres especiales como "\", "[", "]", etc.

A nivel de palabras se ha eliminado lo siguiente:

- Palabras de longitud igual a 1 (“a”, …).
- *Stop Words*. Se ha utilizado el *Corpus* en inglés de la librería “ntlk”

```
from nltk.corpus import stopwords
stopWords = set(stopwords.words('english'))
```

Es interesante apuntar que previamente al tratamiento de palabras se ha aplicado un “*tokenizado*” (punto 2.4.1.5) utilizando la librería “ntlk” y la función “*word_tokenize()*”.

3.2.3 Construcción de datos

Esta tarea implicaría la construcción de nuevos atributos, transformaciones o incluso nuevos registros. En nuestro caso vamos a explicar las técnicas que hemos utilizado para transformar nuestro texto:

- *LowerCasing*. Hemos puesto en minúsculas todos los caracteres existentes.

- Sustitución de contracciones. Hemos utilizado la librería “contractions” para transformar todas las contracciones encontradas para idioma inglés (por ejemplo, *aren’t* por *are not*).

- Sustitución de dígitos por su expresión análoga en números.

3.2.4 Integración de datos

Esta tarea implicaría la combinación de otras entidades para la generación de más información. En nuestro caso no aplicaría directamente, pero sí que, en apartados posteriores, consideraremos Wordnet como base de datos para *lemmatization* y *“AWD-LSTM”* como modelo de lenguaje base a utilizar con algoritmos ULMFiT.

3.2.5 Formato de datos

En este punto incluimos una tarea de “desordenación” de los datos origen aleatoria. De esta forma no estamos ligados a posibles ordenaciones de origen que puedan afectar a nuestros algoritmos.
Utilizamos la función “*shuffle()*” de la librería “*sklearn.utils*”, pasando como parámetro un entero al azar que nos asegure una “desordenación” idéntica. De esta manera obtendremos los mismos resultados en la evaluación de los modelos ML.

```
from sklearn.utils import shuffle # to shuffle the data
```

3.3 Modelado de datos

En esta fase se seleccionarán los modelos a aplicar y se realizarán pruebas para calibrar adecuadamente los parámetros.
Dado que hay varias técnicas para solucionar el mismo problema, con requerimientos distintos, es posible que sea necesario dar pasos atrás para la preparación de datos. Por esta razón se prevé un periodo de refinamiento y repetición de las pruebas.

3.3.1 Selección de las técnicas de modelado

Para la selección de los algoritmos o técnicas a utilizar hemos querido plasmar la utilización de técnicas más tradicionales comparadas con otras de más reciente utilización.

Para las primeras hay multitud de trabajos realizados con distintas herramientas, pero hay un par de ellas especialmente utilizadas, por su tradición y estabilidad, para el análisis de sentimiento, como se comenta en [15]: SVM y *Naïve Bayes*. Hay otros trabajos que también retornan muy buenos resultados utilizando *Logistic Regression*, como los realizados por Kanish Shah, Henil Patel, Devanshi Sanghvi & Manan Shah en 2020 [37].
Basándonos en estos datos y por experiencia personal y profesional, en nuestro caso consideraremos ***Multinomial Naïve Bayes*** **(MNB) y** ***Logistic Regression*** **(LR)**.

Para las segundas habría que considerar técnicas basadas en Deep Learning y a técnicas que hasta ahora se aplicaban mayoritariamente al campo de la Visión por Computador, como el *Transfer Learning*. En nuestro caso seleccionaremos el modelo **ULMFiT** para evaluar sus capacidades.

3.3.2 Diseño de pruebas

Punto afectado	Tipo de Prueba
Stemming/Lemmatization	Uso de uno u otro
Vectorización Tf-idf	Parametrización
Modelado MNB	Parametrización
Modelado LR	Parametrización
Modelado ULMFiT	Activación/Desactivación de capas + ejecución de *epochs*

Por cada modelo se obtendrá una matriz de confusión con los resultados obtenidos para su evaluación.

3.3.3 Construcción de los modelos

Dado que las librerías y utilidades en los algoritmos de MNB y LR difieren bastante de las utilizadas para ULMFiT, hemos decidido abordar la construcción de los modelos por dos vías distintas. De ahí que abordemos sus problemáticas en apartados separados.

3.3.3.1 MNB y LR

3.3.3.1.1 Stemming/Lemmatization

Como inicialmente no hay un consenso único para determinar si es mejor una técnica u otra realizaremos pruebas con las dos técnicas y evaluaremos resultados. Para realizar las pruebas utilizamos la librería "*ntlk.stem*".

En el caso de *lemmatization* utilizaremos Wordnet, como extensa y gratuita base de datos del idioma inglés. Para el caso de *stemming* utilizaremos el algoritmo de Porter [38].

Como se puede comprobar en el código anexado la técnica para los dos casos es la siguiente:

- Revisar todas las palabras una a una que tengamos en nuestros textos.
- Encontrar su *lemma/stem* asociado y sustituir la palabra.

En las pruebas realizadas los resultados finales fueron muy parecidos, así como los tiempos de procesado. Finalmente, y tras comprobar algunos resultados, nos decidimos por utilizar *lemmatization*.

3.3.3.1.2 Vectorización

Optamos por utilizar en este caso Tf-idf, y para ello nos ayudamos de la librería de "ntlk.feature_extraction.text":

```
from sklearn.feature extraction.text import TfidfVectorizer
```

En este caso se han utilizado los textos de entrenamiento y test conjuntamente para el aprendizaje del vocabulario (función "*fit*") y posteriormente se ha generado la matriz de términos ("*document-term"*) por separado. Se ha considerado que, a más cantidad de textos, más completo sería el resultado.

En las pruebas se han utilizado los siguientes parámetros:

- *Max_df*: Máximo porcentaje de palabras a no tratar. Las palabras que estén por encima de ese porcentaje en cuanto a frecuencia no se tratarían.

- *Min_df*: Mínimo porcentaje de palabras a no tratar. Ídem que el anterior, pero en el caso de las palabras que menos aparezcan.

- *Ngram_range*: Tratamiento de palabras o binomio de palabras. Se utilizaron unigramas y binomios 2 y 3.

Después de probar con distintos porcentajes de palabras a no tratar y combinándolo con binomios, no se obtuvo una mejoría significativa de la "calidad" de los datos en el diccionario. Se opta finalmente por no usar ningún parámetro opcional y seguir con los valores por defecto.

3.3.3.1.3 Generación de modelos

Multinomial Naïve Bayes

Se utiliza la librería "*sklearn.naive_bayes*" y su función "*MultinomialNB*":

```
from sklearn.naive bayes import MultinomialNB
        model = MultinomialNB(alpha=0.3)
```

Para la evaluación del modelo se han realizado distintas pruebas utilizando el parámetro "*Alpha*". Este parámetro controla el llamado "*smoothing"* (suavizado) de valores categóricos que aplica el algoritmo, siendo un numérico real entre 0 y 1: 0 no aplicaría nada y 1 sería el máximo (valor por defecto). Tal como se ha explicado en el apartado dedicado a este algoritmo (2.4.3.3.5), con este concepto se pretende que ninguna probabilidad sea cero, y de esta forma regularizar *Naïve Bayes*.

Logistic Regression

Se utiliza la librería "*sklearn.linear_model*" y su función "*LogisticRegression*":

```
from sklearn.linear model import LogisticRegression
```

Para la evaluación del modelo se han probado dos de sus parámetros fundamentalmente:

- *"C"*: Es un parámetro que controla la "regularización", por la cual se intenta controlar el sobre entrenamiento, y que también se aplica en otros algoritmos basados en regresión. Su valor es un número real entre 0 y 1, siendo 0 la máxima regularización aplicada y 1 la mínima (valor por defecto).

- *"Multi_class"*: Si la opción es "*ovr*" se aplica un algoritmo para encontrar valores binarios. Aplicando sin embargo la opción "*multinomial*" se aplica una distribución entre los valores posibles, aunque el resultado final sea 0 ó 1.

3.3.3.2 ULMFiT

Para realizar las pruebas de los modelos con este tipo de técnicas nos basaremos en la librería *"Fast.ai"*, que ofrece una serie de utilidades de muy fácil aplicación.
Las técnicas de utilización se basan en las descripciones realizadas en la web de Fastai[4]. Esta metodología de utilización se basa en una serie de pasos: selección de un lenguaje preentrenado y ajuste de nuestros datos (*"fine-tuning"*) utilizando el lenguaje preentrenado (*"Language Model"*), y finalmente ajustar nuestro clasificador para la tarea de análisis de sentimiento ("Clasificador").

Las librerías que se utilizaran son:

```
import fastai
from fastai.text import *
from fastai.callbacks import *
```

Para empezar, partiremos de nuestros datos ya preprocesados (realizados ya en apartados anteriores), lo que constituye siempre una buena práctica, todo y que estas librerías ya realizan un depurado y transformación de información.

3.3.3.2.1 Language Model

Dado que estamos utilizando librerías de tratamiento específico, hemos de "transformar" nuestra información a un tipo determinado de dato que se usa en *"Fast.ai"*: *"DataBunch"*.

En nuestro caso hemos agrupado la información de entrenamiento y test dentro del objeto de tipo *"DataBunch", "data_lm"*. Hemos de considerar que esta función ya realiza operaciones de preprocesado y transformación de información, como puede ser el hecho del marcado de determinados elementos dentro del texto (por ejemplo *"xxbos"* representa el inicio de texto).

text	target
xxbos match 1 : tag team table match bubba ray and spike dudley vs eddie guerrero and chris benoit bubba ray and spike dudley started things off with a tag team table match against eddie guerrero and chris benoit . according to the rules of the match , both opponents have to go through tables in order to get the win . benoit and guerrero heated up early on by	1
xxbos the premise of this movie has been tickling my imagination for quite some time now . we have all heard or read about it in some kind of con - text . what would you do if you were all alone in the world ? what would you do if the entire world suddenly disappeared in front of your eyes ? in fact , the last part is actually	1
xxbos 8 simple rules for dating my teenage daughter had an auspicious start . the supremely - talented tom shadyac was involved in the project . this meant that the comedy would be nothing less of spectacular , and that is exactly what happened : the show remains one of the freshest , funniest , wittiest shows made in a very long time . every line , facial expression ,	1
xxbos the vigilante has long held a fascination for audiences , inasmuch as it evokes a sense of swift , sure justice ; good triumphs over evil and the bad guy gets his deserts . it is , in fact , one of the things that has made the character of dirty harry callahan (as played by clint eastwood) so popular . he carries a badge and works	1
xxbos warning : spoilers galore ! tim burton remaking this sui generis movie is about as sensible as remaking psycho oh , that is right , some idiot already did that i rest my case . movie opens with chimpnaut blundering a simulation , proving he is not that smart from the outset . marky mark appears in shot without his characteristic underpants showing , then is turned down by	0

Una vez la información esté ingestada podemos proceder al modelado. Para ello seleccionamos el modelo preentrenado *"AWD-LSTM"* [5].

[4] https://docs.fast.ai/tutorial.text.html
[5] https://medium.com/ai%C2%B3-theory-practice-business/awd-lstm-6b2744e809c5

Con el modelo seleccionado podemos comenzar a entrenar el modelo. En este punto nos será necesario encontrar previamente una tasa de aprendizaje (del inglés *"learning rate"*) adecuada para que el ajuste del modelo sea óptimo. Una función muy interesante para esta tarea, y que utilizaremos recursivamente, es "*lr_find()*".

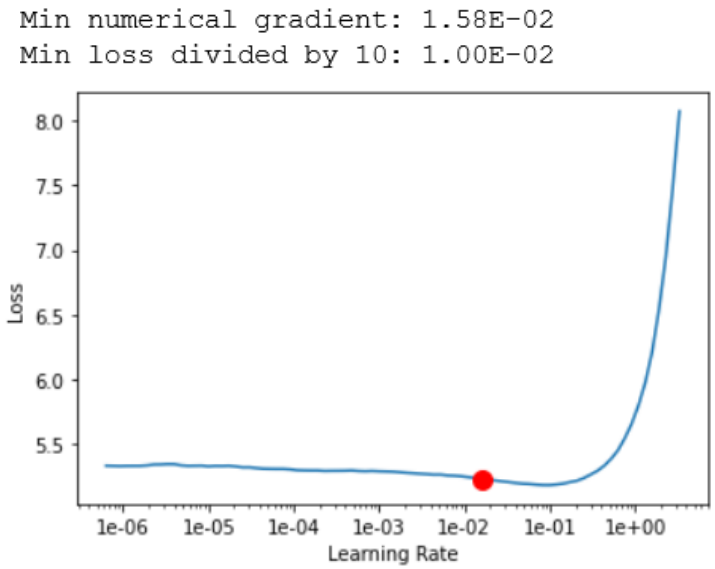


Figura 22. Resultados prueba funcion "*lr_find()*".

Para una selección óptima se recomienda utilizar el *"learning rate"* que está en el punto de máxima inclinación de caída en la gráfica. Para ayudarnos utilizamos la función *"min_grad_lr"* para seleccionar este punto. En nuestro caso 1,58E-02.

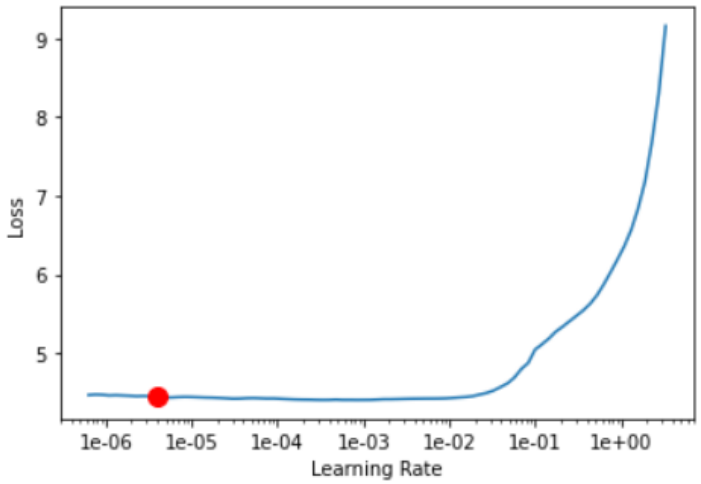
Otro de los parámetros a tener en cuenta es el llamado *"momentum"*. En experimentos previos realizados por Leslie N.Smith, 2018 [39] descubrió que la disminución del *"momentum"* conducía a mejores resultados. De hecho, en la práctica se recomienda utilizar valores como 0,85 o 0,95, y disminuir del mayor a menor cuando se aumenta el *"learning rate"*, para luego volver al *"momentum"* más alto a medida que el *"learning rate"* disminuye. Esta metodología de aprendizaje cíclica se aplica por cada pasada o ajuste de modelo (*"epoch"*) y, para nuestro caso, "fast.ai" tiene una función que lo automatiza: "*fit_one_cycle()".* Esta función permite realizar un número de *"epochs"* determinadas, de forma que a partir de "learning rates" y "momentum" iniciales encontrar los mejores valores después de cada *"epoch"*, seleccionando finalmente la que mejor resultado retorne.

De momento empezamos con una sola pasada para probar resultados iniciales:

epoch	train_loss	valid_loss	accuracy	time
0	4.612718	4.459847	0.251993	09:06

Como se puede apreciar los resultados son pobres. Probablemente el modelo utilizado no sea el mejor para textos como los que estamos procesando. Independientemente de esto se pueden mejorar resultados activando y desactivando capas del modelo (*"freeze"/"unfreeze"*).
Vamos a realizar una prueba activando todas las capas ("*unfreeze*"):

```
Min numerical gradient: 3.98E-06
Min loss divided by 10: 3.31E-05
```



epoch	train_loss	valid_loss	accuracy	time
0	4.377138	4.426826	0.254821	10:14

Los resultados mejoran muy poco. Probablemente deberíamos ejecutar más "*epochs"* y ver si con más pases se observa mejoría.

Independientemente de estos primeros resultados obtenidos, para un ajuste óptimo del modelo se recomienda descongelar capas de forma gradual y ejecutar *"epochs"* cada vez. De esta forma se deberían observar mejorías en los resultados, puesto que el modelo ajustaría mejor al "aprender" de anteriores *"epochs"*.

Otra técnica (y que nos ha dado buenos resultados) en la aplicación del *"learning rate"* es aplicar diferentes valores a diferentes grupos de capas. Esto se consigue utilizando la función *"slice()"* dentro de la función *"fit_one_cycle()"* para asignar los *"learning rate"*, de forma que, por ejemplo:

slice(start = 1e-5, stop = 1e-4)

, asignaría 1e-5 a la capa más alta y 1e-4 a la capa más baja. El resto de las capas tendrían valores entre 1e-5 y 1e-4.

3.3.3.2.1 Clasificador

Una vez ya hemos hecho lo posible entrenando el modelo base seleccionado, entramos en la parte de clasificación aplicada a nuestra problemática de análisis de sentimiento.
Nuevamente hemos de ingestar la información para la clasificación. Para ello utilizaremos la función *"TextClasDataBunch()"*. A partir de esto ya podemos proceder al modelado de nuestros datos.

Ahora se trataría, como se ha hecho en el anterior apartado, de ajustar el modelo siguiendo los pasos ya vistos: encontramos el mejor "learning rate", activamos/desactivamos capas y aplicamos el modelo.

En el siguiente apartado evaluaremos los resultados.

3.3.4 Evaluación de resultados

Para la evaluación de resultados de MNB y LR utilizamos la librería de "*sklearn.metrics"* y las funciones "*confusion_matrix()*" y "*classification_report()*":

```
from sklearn.metrics import confusion_matrix, classification_report
```

La función "*predict()*" nos calculará las predicciones efectuadas por el modelo y nos permitirá utilizar luego obtener la matriz de confusión.
Con la función "*score()*" hemos pretendido obtener el valor de "*accuracy*" con una precisión de dos decimales, para así tener esta medida de evaluación con más detalle y comparar entre ejecuciones.

Para la obtención de las medidas de evaluación se utilizan los valores de *test* contra el modelo generado con los datos de entrenamiento.

Por otro lado, para la evaluación de resultados de ULMFiT utilizamos las funciones de la librería "fastai", incluida la propia función de modelado "*"fit_one_cycle().*

Notar que para ULMFiT se ha utilizado *Google Colab* con GPU, ya que los tiempos de procesado necesario para *"Fast.ai"* son altos. No se han podido utilizar procesadores de tipo TPU ya que no son compatibles con la librería Python "fastai".

Multinomial Naïve Bayes

Aplicando las distintas pruebas señaladas en el apartado 3.3.2 se han obtenido unos resultados muy parecidos, con una exactitud entorno al 86%.
El tiempo de generación del modelo es prácticamente inmediato.

Cuadro de pruebas realizadas:

Stem/Lemmat. (S/L)	***Vectorization***	***Parameters***	***Accuracy(%)***
L	-	alpha=0.3	86,90%

S	-	alpha=0.3	86,30%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	alpha=0.3	85,58%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	alpha=0.3	85,24%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(2,3)	alpha=0.3	65,78%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(2,3)	alpha=0.3	67,32%
L	-	alpha=0.5	86,92%
S	-	alpha=0.5	86,48%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	alpha=0.5	85,60%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	alpha=0.5	85,26%
L	-	alpha=0.7	85,60%
S	-	alpha=0.7	85,30%

Los mejores resultados obtenidos finalmente:

```
######################
MULTINOMIAL NAIVE BAYES
######################
Total time.... 0.05 secs
Confusion matrix....
[[2185  310]
 [ 344 2161]]
Accuracy.... 86.92%
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      2495
           1       0.87      0.86      0.87      2505

    accuracy                           0.87      5000
   macro avg       0.87      0.87      0.87      5000
weighted avg       0.87      0.87      0.87      5000
```

Logistic Regression

Los peores resultados se obtenían modificando la parametrización de “C” o regularización, por lo que se ha optado por no aplicarla. Sin embargo, con la aplicación del parámetro (*multi_class=’multinomial’*) se han conseguido unos valores de exactitud algo superiores al 90%.
De la misma forma se ha observado que el parámetro aplicado aumenta el tiempo de generación del modelo, pasando de 1.5seg a algo más de 5seg aproximadamente.

Cuadro de pruebas realizadas:

Stem/Lemmat. (S/L)	***Vectorization***	***Parameters***	***Accuracy(%)***
L	-	multi_class='multinomial'	90,02%
S	-	multi_class='multinomial'	89,74%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	multi_class='multinomial'	87,94%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	multi_class='multinomial'	87,48%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(2,3)	multi_class='multinomial'	67,90%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(2,3)	multi_class='multinomial'	69,32%
L	-	C = 0.5	89,34%
S	-	C = 0.5	89,14%
L	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	multi_class='multinomial' C = 0.5	87,96%
S	analyzer = "word", max_df=0.9,min_df=0.01, ngram_range=(1,3)	multi_class='multinomial' C = 0.5	87,76%
L	-	multi_class='multinomial' C = 0.3	87,90%
S	-	multi_class='multinomial' C = 0.3	87,84%

Estos han sido los mejores resultados obtenidos:

```
######################
LOGISTIC REGRESSION
######################
Total time.... 5.31 secs
Confusion matrix....
[[2223  272]
 [ 227 2278]]
Accuracy.... 90.02%
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      2495
           1       0.89      0.91      0.90      2505

    accuracy                           0.90      5000
   macro avg       0.90      0.90      0.90      5000
weighted avg       0.90      0.90      0.90      5000
```

ULMFiT

La mejor ejecución obtenida nos retorna una exactitud del 92,58% realizando distintas activaciones de capas y ejecuciones de *epochs*.

Hay que comentar que los tiempos de ULMFiT aproximadamente han sido los siguientes (utilizando GPU *Google Colab*):

- Ingesta de valores → 45min aprox.
- Búsquedas de "*learning rate*" → Menos de 1min.
- Entrenamientos por *epoch* → Entre 4 y 10 min según las capas activas (cuantas más capas activas más tiempo de proceso).

Cuadro de pruebas realizadas:

	Test#	***Learning Rate***	***Momentum***	***Layers unfroz./Epochs***	***Accuracy(%)***
Language Model	#1	6,31E-07	0,95-0,85	0/1	20,91%
	#2	2,29E-06	0,8-0,7	0/1	21,03%
		3,31E-04	0,9-0,8	*All*/2	27,11% 27,76%
	#3	1,58E-02	0,8-0,7	0/1	25,19%
		3,98E-06	0,8-0,7	*All*/1	25,48%
	#4	1.45E-05	0,8-0,7	0/1	21,67%
		1.74E-03	0,8-0,7	*All*/1	26,87%
Classifier	#1	1,45E-03	0,9-0,8	1/1	89,60%
		6,31E-07	0,9-0,8	2/3	89,66% 89,26% 89,20%
		6,31E-07	0,9-0,8	3/1	89,70%
		6,31E-07	0,8-0,7	*All*/1	92,42%
	#2	1.32E-02	0,8-0,7	1/1	88,36%
		1,32E-06	0,8-0,7	2/3	89,10% 89,34% 89,70%
		1,58E-06	0,8-0,7	3/1	91,32%
	#3	*slice*(1E-02/(2,6**4), 1E-02)	0,8-0,7	1/1	87,70%
		slice(1E-02/(2,6**4), 1E-02)	0,8-0,7	2/1	91,48%
		slice(5E-03/(2,6**4), 5E-03)	0,8-0,7	3/1	92,56%
		slice(1E-03/(2,6**4), 1E-03)	0,8-0,7	*All*/1	92,58%
	#4	6.92E-04	0,8-0,7	1/1	88,66%
		6.31E-07	0,8-0,7	2/1	89,26%

		slice(5E-03/(2.6**4), 5E-03)	0,8-0,7	3/1	92,54%
		slice(1E-03/(2.6**4), 1E-03)	0,9-0,8	*All*/1	92,86%

A continuación, mostramos los mejores resultados obtenidos en el “Test#4” paso a paso, de forma que se puede observar la progresiva evolución de los resultados. La misma función *“fit_one_cycle()”* nos retorna el porcentaje de *“accuracy”* por lo que podemos ver esa evolución. Finalmente se mostrarán los resultados finales con una matriz de confusión para tener los números completos y una serie de *“tests”* realizados con texto libre.

---------- Language Model----------
(#4;1 epoch, Learning Rate=1,45E-05)

epoch	train_loss	valid_loss	accuracy	time
0	5.157081	5.058714	0.216703	16:24

(#4;1 epoch, todas las capas activadas, Learning Rate=1,74E-03))

epoch	train_loss	valid_loss	accuracy	time
0	4.439065	4.309174	0.268732	18:42

---------- Clasificador ----------
(#4;1 epoch, una capa activada, Learning Rate=6.92E-04)

epoch	train_loss	valid_loss	accuracy	time
0	0.320593	0.274743	0.886600	03:26

(#4;1 epoch, dos capas activadas, Learning Rate=6.31E-07)

epoch	train_loss	valid_loss	accuracy	time
0	0.301708	0.272888	0.892600	04:00

*(#4;1 epoch, tres capas activadas, Learning Rate=slice(5E-03/(2.6**4),5E-03))*

epoch	train_loss	valid_loss	accuracy	time
0	0.210100	0.191758	0.925400	06:28

(#4; 1 epoch, todas activadas, Learning Rate= slice(1E-03/(2.6**4),1E-03)*)*

epoch	train_loss	valid_loss	accuracy	time
0	0.176655	0.179927	0.928600	08:18

---------- *Resultados finales* ----------

```
# get predictions
preds, targets = learn4.get_preds()

predictions = np.argmax(preds, axis = 1)
pd.crosstab(predictions, targets)
```

col_0	0	1
row_0		
0	2316	178
1	179	2327

---------- Testing ----------

```
#Testing examples
print(learn4.predict("huge film"))

print(learn4.predict("so bad but i like it"))

print(learn4.predict("Last but not the least; the androgynous beauty of the sexy Renée Soutendijk "+
 "perfectly fits to her role of a woman that attracts a gay writer. My vote is eight."))

print(learn4.predict("This film takes you on one family's impossible journey, and makes you feel " +
 "every step of their odyssey. Beautifully acted and photographed, heartbreakingly real. Its last "+
 "line, with its wistful hope, is one of the more powerful in memory"))

print(learn4.predict("I saw this in the summer of 1990. I'm still annoyed by how bad this movie is "+
 "in 2001.<br /><br />Implausible plot. You'd have to be a child to think this could happen."+
 "<br /><br />I'm just really annoyed by it. Don't see this."))

print(learn4.predict("Nothing great here but a nicely acted story about an abused deaf wife (Fonda) "+
 "of a small time crook (Bochner)who gets involved with one of her husband's plans and his mistress. "+
 "Sutherland and Weber are cops drawn into what turns out to be a unmysterious murder investigation "+
 "and the story just flows along."))
```

```
(Category tensor(1), tensor(1), tensor([0.0680, 0.9320]))
(Category tensor(0), tensor(0), tensor([0.5725, 0.4275]))
(Category tensor(1), tensor(1), tensor([0.0166, 0.9834]))
(Category tensor(1), tensor(1), tensor([3.8348e-04, 9.9962e-01]))
(Category tensor(0), tensor(0), tensor([0.9669, 0.0331]))
(Category tensor(1), tensor(1), tensor([0.0269, 0.9731]))
```

4. Desarrollo *Front-end*

Como ya se contempló en el inicio del proyecto, se requiere la generación de una utilidad web que permitiera a un usuario realizar pruebas de los modelos calculados en el punto anterior, evaluando su calidad. Por otro lado, de esta forma, nos independizamos de la arquitectura de la máquina utilizada para realizar las pruebas, puesto que solo sería necesario contar con un navegador web instalado.

Para la realización de esta utilidad web se utilizarán las siguientes tecnologías:

- ***HTML***. Acrónimo de "*HiperText Markup Language",* es un lenguaje para el desarrollo de páginas web basado en etiquetas.
- ***CSS***. Creado para la simplificación de HTML su utilidad es la de definir hojas de estilo en cascada. Permite la definición de estilos sin repetir código en HTML y de esta forma simplificar su uso.
- ***Bootstrap***. *Framework* de trabajo utilizado mayoritariamente para la coordinación de las tres anteriores tecnologías comentadas. Se compone de un juego de hojas de estilo y de una librería de *javascript*, mediante las cuales genera un potente marco para la creación de nuestras páginas web.
- ***PHP***. Es un lenguaje para el desarrollo de aplicaciones en la web. Se ha convertido en una de las utilidades de más popularidad para la creación de páginas dinámicas, entre otras cosas por su sencillez, la posibilidad de aplicarse de forma embebida en código HTML y su capacidad de funcionamiento en multiplataforma.

Para obtener información sobre el código fuente generado consultar el punto "8.Anexos" de este documento.

4.1 Requerimientos previos

Con la idea de no generar una aplicación demasiado compleja, a nivel funcional tendríamos los siguientes puntos:

- Posibilidad de seleccionar el modelo NLP a probar.
- Introducción de texto libre para la evaluación de sentimiento.
- Visualización de los resultados: clasificación y porcentaje de probabilidad.

Por otro lado, a nivel de infraestructura técnica los requerimientos de funcionamiento de la aplicación serian:

- La aplicación tiene que ser capaz de invocar procesos *Python*, los cuales serán los encargados de cargar y evaluar los modelos previamente calculados.

- La aplicación tiene que poderse ejecutar en cualquier máquina que tenga un navegador web de uso común (*IExplorer*, *Firefox*, *Chrome*, …).

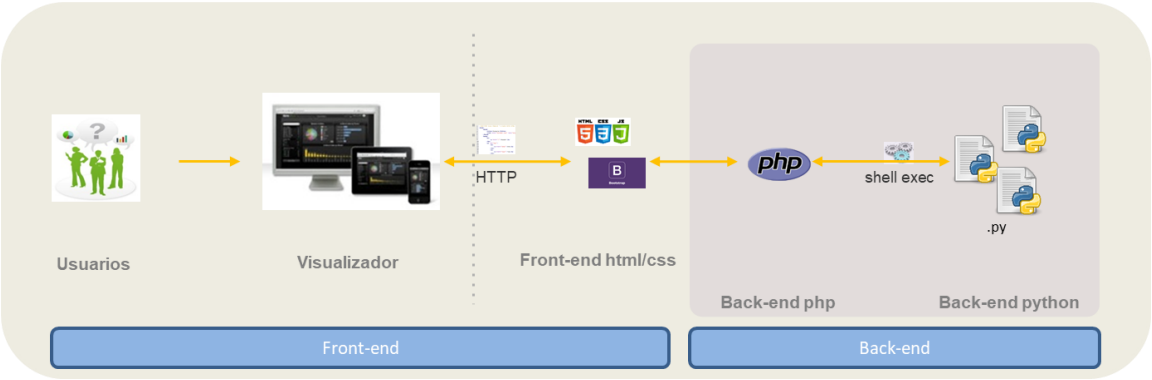


Figura 23. Esquema de flujos.

4.2 Diseño preliminar

El diseño de la pantalla contemplará básicamente dos zonas:

- Parte izquierda: donde encontramos una lista desplegable para seleccionar el modelo y el botón de cálculo del modelo
- Parte derecha: encontramos una zona de texto de entrada (arriba) y otra de salida donde retornar los resultados (debajo).

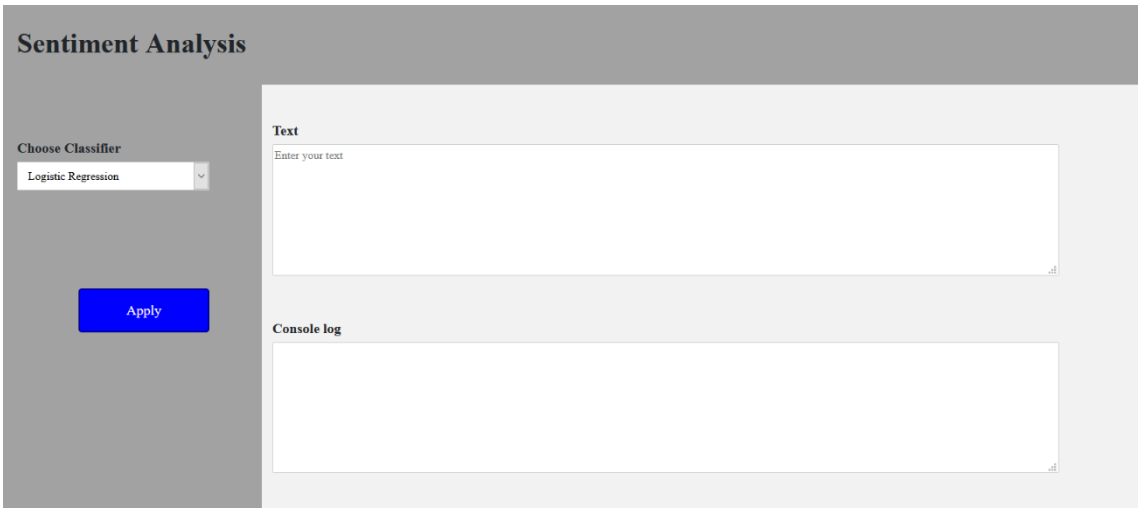


Figura 24. *Front-end web*.

4.3 Desarrollo

Todo y que estamos en la parte de *front-end* visual de información en una aplicación de este estilo siempre hay una parte *back-end* relacionada, que es la encargada de gestionar esa interfaz y ejecutar aquellos procesos que permitan obtener unos resultados.

Podemos por tanto separar nuestro desarrollo relacionado con la interfaz en dos partes:

- Una parte *front-end* definida con código *HTML* y que utiliza una hoja de estilos CSS para la definición de tipos y tamaños de letra, colores, tamaños de objetos, etc.

- Por otro lado, encontramos la parte *back-end* de la herramienta interfaz. Esta se compone de un apartado PHP que se encarga de la gestión de la lista desplegable y de la llamada a los ficheros Python con extensión .py.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Sentiment Analysis</title>
  <link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <script src="https://code.jquery.com/jquery-3.5.0.js"></script>
  <script src="bootstrap/js/bootstrap.min.js"></script>
<form action="index.php" method="POST" title>
  <link type="text/css" rel="stylesheet" property="stylesheet" href="css/newstyle.css">
  <div class="container-fluid">
  <div class="row">
  <div class="col-md-12 well">
  <label class="title">Sentiment Analysis</label>
  </div>
  </div>
  <div class="row">
  <div class="col-md-3 well">
  <div class="elem-group">
    <label for="classifier">Choose Classifier</label>
    <select id="classifier" name="classifier" required>
        <option value="LR" <?php if(@$_POST['classifier'] == 'LR') { echo 'selected = \"selected\"'; } ?>>Logistic Regression</option>
        <option value="MNB" <?php if(@$_POST['classifier'] == 'MNB') { echo 'selected = \"selected\"'; } ?>>Multinomial Naive Bayes</option>
        <option value="ULM" <?php if(@$_POST['classifier'] == 'ULM') { echo 'selected = \"selected\"'; } ?>>ULMFiT</option>
    </select>
  </div>
  <br><br><br>
  <button type="submit" name="Apply">Apply</button>
  </div>
  <div class="col-md-8 well">
  <div class="elem-group">
      <label for="txt">Text</label>
      <textarea id="txt" name="txt" required placeholder="Enter your text"></textarea> <br><br><br>
      <label for="log">Console log</label>
      <textarea readonly><?php if(isset($_POST['Apply'])){$message = escapeshellarg ($_POST['txt']);
                                                          $model=$_POST['classifier'];
                                                          if ($model == 'LR') {
                                                            $output = shell_exec("python3 ./Python/TFG_jchulilla_PEC3_TEST_LR.py $message");}
                                                          else if ($model == 'ULM') {
                                                            $output = shell_exec("python3 ./Python/TFG_jchulilla_PEC3_TEST_ULM.py $message");}
                                                          else if ($model == 'MNB') {
                                                            $output = shell_exec("python3 ./Python/TFG_jchulilla_PEC3_TEST_MNB.py $message");}
                                                          else {$output = "ERROR";}
                                                          print_r($message);
                                                          echo("").PHP_EOL;
                                                          echo("").PHP_EOL;
                                                          print_r("Results.....");
                                                          echo("").PHP_EOL;
                                                          print_r($output);}
                                                          ?>
      </textarea>
  </div >
</div >
</div >
</div >
</form>
</body>
</html>
```

Figura 25. Código *front-end*.

En el siguiente punto se explica la parte Python que nos permite evaluar los modelos generados previamente.

4.3 Integración *back-end*

En la parte de integración con Python, necesitamos algún proceso que se encargue de cargar los modelos generados en los procesos del apartado 3 “Desarrollo Back-end”, evaluar el texto pasado por parámetro por la capa PHP y retornar resultados.

En nuestro caso se ha decidido generar un fichero .py separado por cada modelo para mayor comodidad.

```python
import pickle
import io
import sys

path='./Python'

loaded_model = pickle.load(open(path+'/modelLR.pkl', 'rb'))
vectorizer = pickle.load(open(path+'/tfidf_vectorizer.pkl', 'rb'))

#Loading text
text=sys.argv[1]

#Saving result
sentiment=loaded_model.predict(vectorizer.transform([text]))[0] #0-negative/1-positive
[number1,number2]=loaded_model.predict_proba(vectorizer.transform([text]))[0,]

if sentiment==0:
    print("--> Sentiment: Negative")
    print("--> Probability: "+str(round(number1*100,2))+'%')
else:
    print("--> Sentiment: Positive")
    print("--> Probability: "+str(round(number2*100,2))+'%')
```

Figura 26. Código Python "TFG_jchulilla_PEC3_TEST_LR.py"

```python
import io
import fastai
from fastai.text import *
import sys,os

#Turning off stdout,stderr
_stderr = sys.stderr
_stdout = sys.stdout
null = open(os.devnull,'wb')
sys.stdout = sys.stderr = null

#Loading model
path='./Python'
learn = load_learner(path,'learn.pkl')

#Turning on stdout,stderr
sys.stderr = _stderr
sys.stdout = _stdout

#Loading text
text=sys.argv[1]

#Saving result
result = str(learn.predict(text))

sentiment=result[17] #0-negative/1-positive
numbers=result[41:-1].replace(')','').replace(']','').replace(',','').replace(' ','\n')
buffer=io.StringIO(numbers)
number1=float(buffer.readline())
number2=float(buffer.readline())

if sentiment=="0":
    print("--> Sentiment: Negative")
    print("--> Probability: "+str(round(number1*100,2))+'%')
else:
    print("--> Sentiment: Positive")
    print("--> Probability: "+str(round(number2*100,2))+'%')
```

Figura 27. Código Python "TFG_jchulilla_PEC3_TEST_ULM.py"

Estos ficheros operan básicamente de la misma manera, todo y que usan librerías distintas para el tratamiento del modelo generado según el algoritmo.

Los ficheros .py de los algoritmos LR y MNB son prácticamente iguales, al haber utilizado las librerías “sklearn” y “pickle” para la generación y guardado de los modelos respectivamente. Por otro lado, ULMFiT utiliza la librería “fastai” para ambas cosas. Operaciones que se realizan:

- Carga de los modelos; ficheros con extensión pkl. (los modelos LR y MNB además cargan el vectorizador “tfidf_vectorizer.pkl”)
- Ejecución de los modelos para el texto pasado por parámetro.
- Generación de los resultados:
    - “*Sentiment*”: positivo o negativo
    - “*Probability*”: porcentaje de acierto de la clasificación.

4.3 Pruebas

Una vez se consigue que el proceso funcione correctamente se han realizado diversas pruebas para comprobar la calidad de las clasificaciones. Por un lado, hemos introducido texto libre, no siempre aplicable a un lenguaje relacionado con opiniones de películas, y por otro lado hemos aprovechado textos ya etiquetados del fichero “Valid.csv” (presente en la web de Kaggle de donde hemos recogido nuestros *dataset* para el entrenamiento de los modelos).
En general los resultados obtenidos han sido bastante lógicos, pero se nota en exceso que el entrenamiento se ha basado en un *dataset* con un lenguaje concreto, y todo aquello que se aleje de esta tipología no asegura unos resultados aceptables.
Por ejemplo, utilizando la palabra “*good*” en el caso de MNB el resultado es positivo, pero con una probabilidad en el borde del 50%.

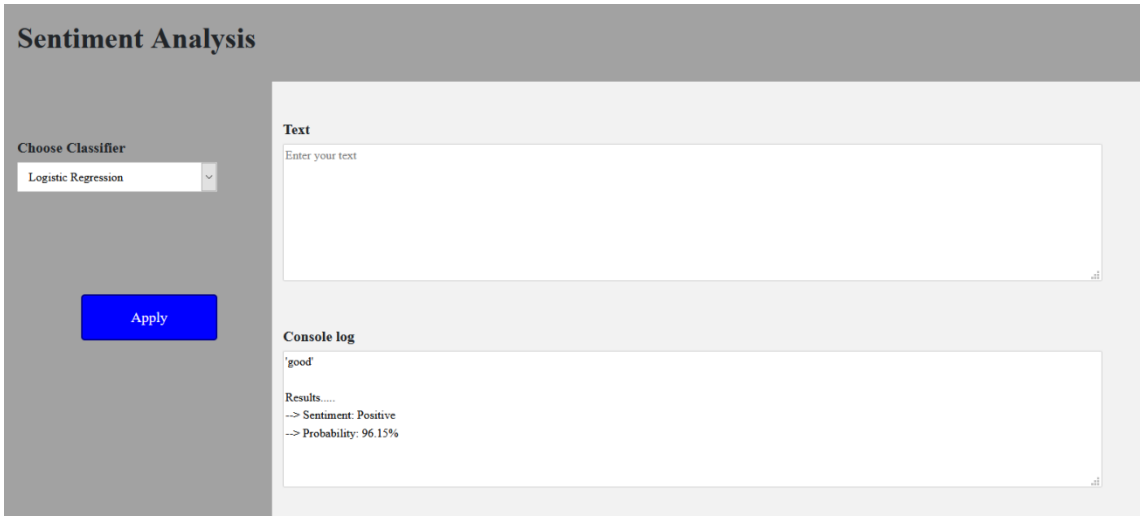


Figura 28. Prueba texto "good" para LR.

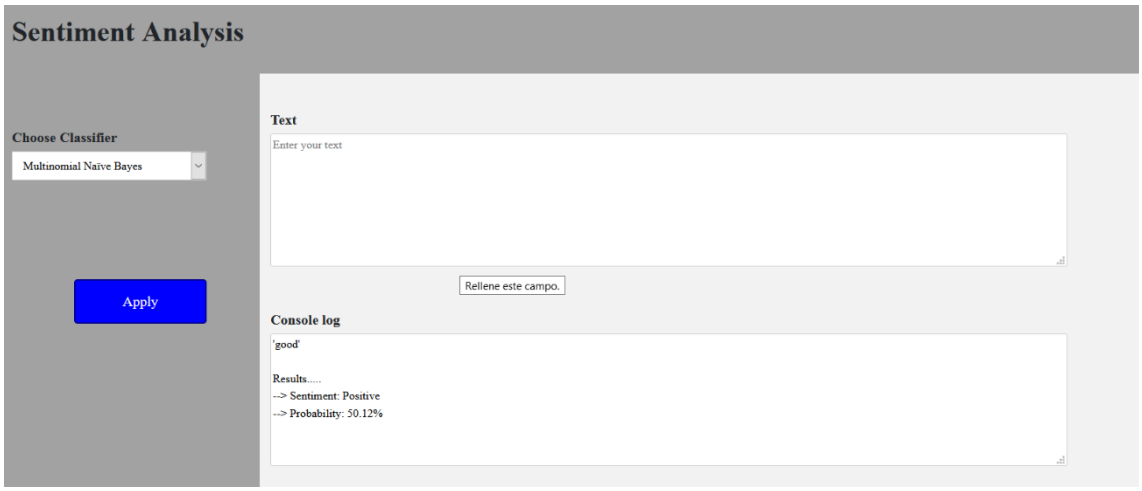


Figura 29. Prueba texto "good" para MNB.

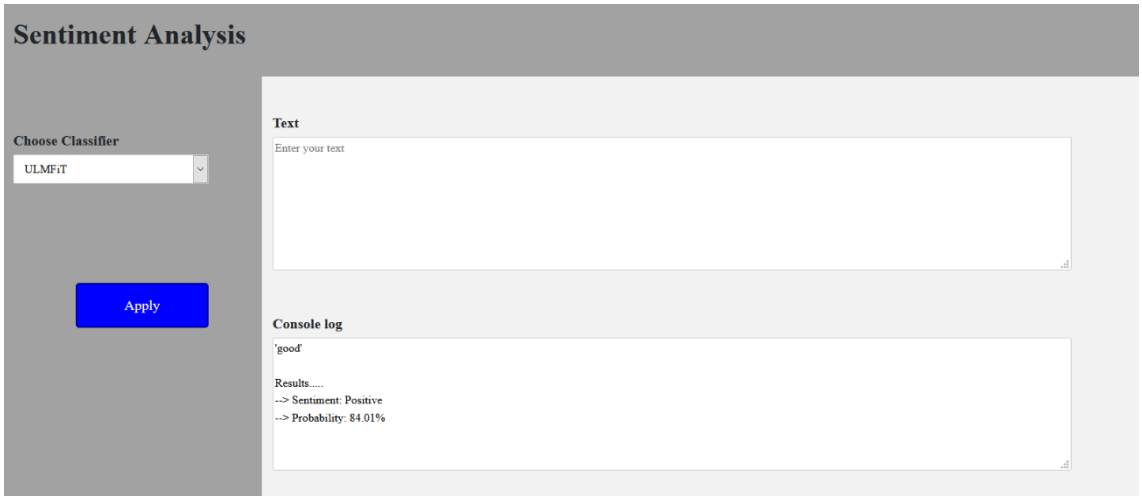


Figura 30. Prueba texto "good" para ULMFiT.

Sin embargo, para el caso de un texto como el siguiente:

> *“This sequel to Problem Child is just as bad as the first one. It still teaches kids that it's O.K. to be bad. It's impossible for me to recommend this movie to anyone.”*

, extraído de “Valid.csv” y de sentimiento claramente negativo, los resultados son más concluyentes.

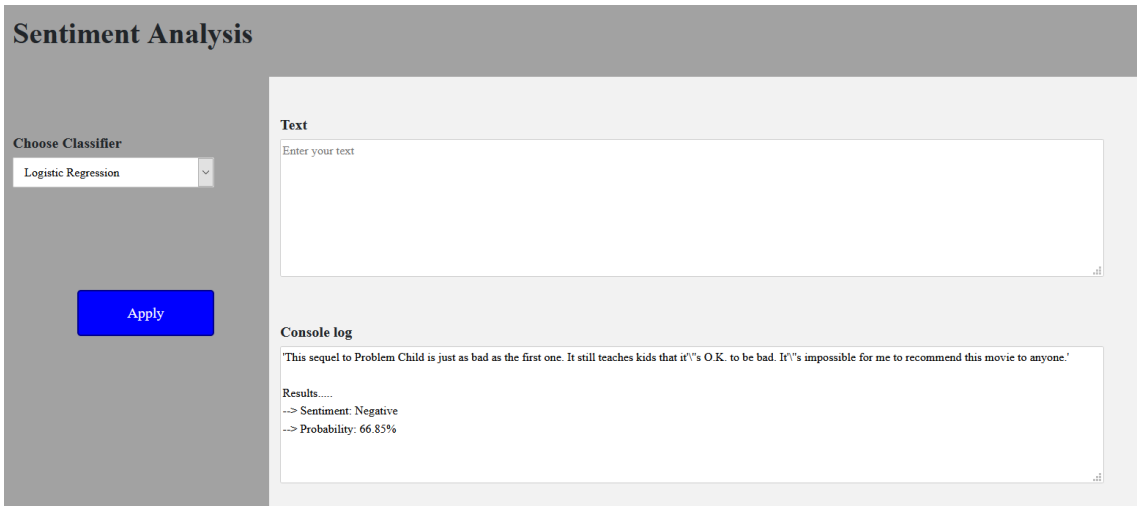


Figura 31. Prueba texto de "Valid.csv" para LR.

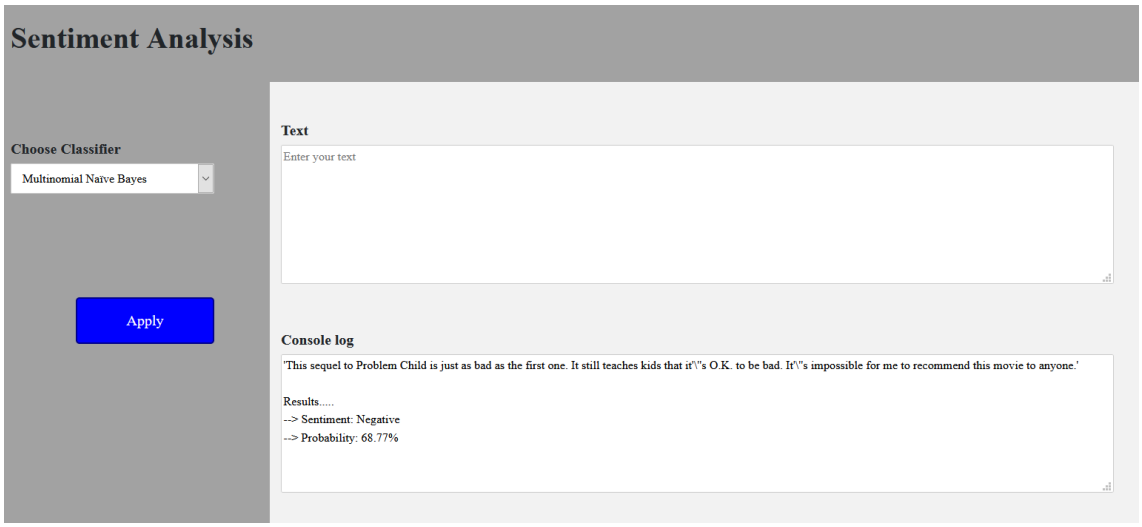


Figura 32. Prueba texto de "Valid.csv" para MNB.

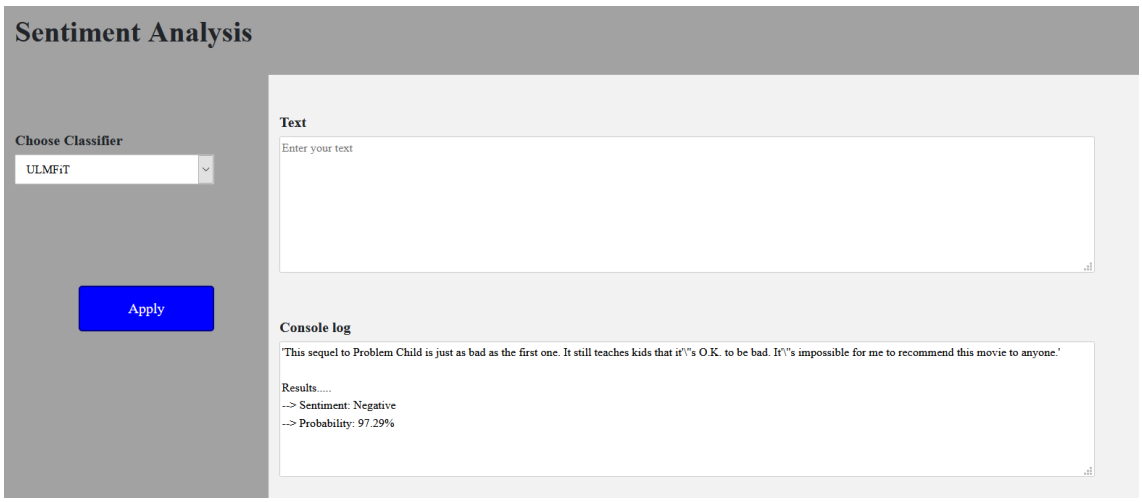
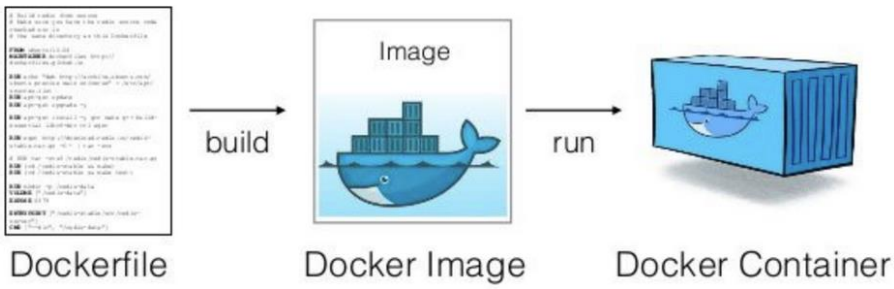


Figura 33. Prueba texto de "Valid.csv" para ULMFiT.

4.3 Despliegue

Inicialmente se contemplaba la opción de desplegar la aplicación web en Cloud, concretamente utilizando las herramientas de *Azure Cloud App Platform*. Por problemas de licencias/costes se ha descartado esta opción. Finalmente se ha optado por realizar el despliegue utilizando imágenes/contenedores "Docker" (https://www.docker.com/). Docker es una herramienta que automatiza el despliegue de aplicaciones dentro de contenedores de software, actuando como máquinas virtuales basadas en un *kernel* Linux. Esto nos permite compartir y desplegar la aplicación en local, pero por el contrario obliga a instalar la herramienta en el equipo.

La generación de la imagen Docker ("build") se basa en un fichero con nomenclatura "DockerFile", en el que se definen básicamente las aplicaciones y los ficheros que contendrá la imagen. En nuestro caso la base es una imagen con la aplicación servidor "*Apache*", a la cual se ha instalado el intérprete de Python y sus librerías necesarias, y se han copiado los ficheros de la página web y los ejecutables con extensión .py. Una vez generada la imagen se ha compartido en el repositorio *Cloud* "Docker Hub" con el tag "jchulilla/tfg".
Para más información sobre el despliegue y los ficheros utilizados consultar el punto "8.Anexos" de este documento.



El comando "docker" para bajar la imagen en el equipo seria:

```
>docker pull jchulilla/tfg
```

Una vez bajada ya estaría disponible para ejecutar ("run") en un contenedor "docker", utilizando el comando:

```
>docker run -d -p 9090:80 jchulilla/tfg
```

Y a partir de aquí ya podemos utilizar nuestro navegador web para abrir la aplicacion desde http://localhost:9090 (9090 es el puerto que hemos redireccionado al ejecutar la imagen, aunque se puede usar el que se quiera siempre y cuando esté libre para su uso en el equipo local).

5. Conclusiones

La comprensión del idioma es un desafío para las computadoras. Los sutiles matices de la comunicación que los niños pequeños pueden entender no

pueden ser procesados por las máquinas hoy en día. Aunque las últimas técnicas pueden detectar y replicar patrones de lenguaje complejos, los modelos de aprendizaje automático aún carecen de una comprensión conceptual de lo que realmente significa el lenguaje humano. La comprensión del contexto es un punto que ha derribado barreras dentro del NLP permitiendo un avance significativo en estas técnicas.

Aunque el futuro en estos tiempos es muy cambiante y lleno de riesgos dentro del NLP, esta disciplina está sufriendo una gran evolución de forma muy rápida. La comunidad científica dedicada ha presentado herramientas muy potentes de libre uso que permiten las pruebas y el constante aprendizaje. Dentro de este ámbito nuevos tipos de técnicas están llamadas a sustituir el actual estatus basado en “vectorizaciones”. Herramientas como ELMo, Google BERT, OpenAI GPT, ULMFiT o Facebook PyText representan este cambio.

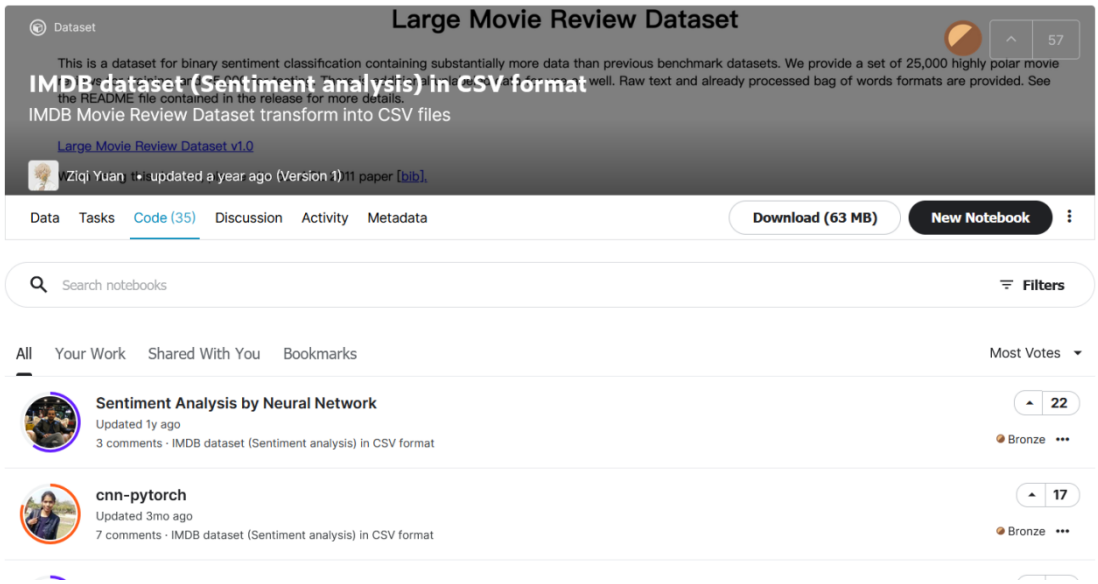
Dentro de este proyecto se ha pretendido por un lado realizar un estado del arte de los sistemas y técnicas de NLP, y por otro la realización de pruebas con casos concretos introduciendo una de las nuevas técnicas expuestas: ULMFiT.

De forma práctica, para nuestro caso, hemos utilizado ULMFiT obteniendo unas tasas superiores al 92%, superando los resultados obtenidos con MNB y LR, a costa eso sí, de un mayor tiempo de proceso. Hay que tener muy en cuenta que los tiempos no son equiparables ya que la CPU/GPU utilizada para MNB y LR no es comparable con la GPU utilizada para ULMFiT. Más allá de esto hemos podido verificar cómo, utilizando una nueva técnica y con una serie de herramientas basadas en una librería de fácil aplicación, se han podido conseguir mejores resultados que con técnicas más “clásicas”. El potencial que tienen este tipo de algoritmos es enorme, y con la aparición de nuevos “diccionarios” que nos permita aprovechar conocimiento permitirá también la mejora en cuanto a precisión. Por esta razón, en nuestra opinión, el coste de recursos justificará su uso, sobre todo teniendo en cuenta que los recursos serán mejores y más baratos en el tiempo. Por tanto, aunque solo sea por el potencial que tiene para nosotros es el claro ganador de esta comparativa. Todo y ello ha de quedar constancia del buen resultado de todas las técnicas en conjunto, llegando a tasas de exactitud alrededor del 90%.

Una de las lecciones aprendidas a base de realizar pruebas es la dependencia al *dataset* que tienen los modelos generados. En este sentido nos hemos encontrado, sobre todo para el caso de ULMFiT, unos tiempos de procesamiento muy elevados y que se debían, en parte, al tamaño de los textos tratados. Una reducción en el número máximo de palabras hubiera optimizado esto, pero nos quedamos con la duda de los resultados que podríamos haber obtenido.

5.1 Comparativa con otros trabajos realizados

En el repositorio Kaggle, del cual hemos obtenido el *dataset*, hemos podido comprobar otras líneas de trabajo realizadas por otros compañeros. En ellos se puede encontrar una gran variedad de aplicaciones de algoritmos, desde los más novedosos a los más clásicos.



Nos hemos centrado en los más votados y comentados, para los cuales se observa una mayor tendencia a utilizar algoritmos de nuevo cuño como pueden ser aquellas que utilizan redes neuronales como base.

A continuación, citamos cuatro trabajos de los encontrados con obtención de resultados bastante similares al nuestro:

- Abdül Meral (Turquía), 2020[6]. Utilizando un vectorizador One-Hot Encoding y Deep Learning con librería "keras" → Precisión sobre el 90%.

- Diksha Bhati (India), 2021[7]. Utilizando un *stemmer* Porter y algoritmo CNN con librería "torch" → Precisión sobre el 88%.

- Avnika Shah (India), 2020[8]. Utilizando un stemmer Porter y un vectorizador Tf-Idf. Algoritmo LR con *6-Fold Cross Validation* y librería "sklearn" → Precisión sobre el 89%.

- Kritanjali Jain (India), 2021[9]. Utiliza BERT con librería "*tensorflow*" → Precisión sobre el 87%.

Como se puede observar existe una buena variedad de trabajos, pero los resultados van un poco en la línea que hemos obtenido en este TFG, y es que las herramientas más novedosas son las que van tomando el relevo, ofreciendo unos resultados mejores. Sin embargo, la diferencia todavía es pequeña y hay

[6] https://www.kaggle.com/abdulmeral/sentiment-analysis-by-neural-network
[7] https://www.kaggle.com/dikshabhati2002/cnn-pytorch
[8] https://www.kaggle.com/avnika22/imdb-perform-sentiment-analysis-with-scikit-learn
[9] https://www.kaggle.com/kritanjalijain/movie-review-sentiment-analysis-eda-bert

que considerar que todavía los recursos a utilizar son más costosos en tiempo y dinero.

5.2 Líneas de trabajo a futuro

Los resultados obtenidos a la hora de comparar texto libre pueden no ser todo lo concluyentes que nos ha ofrecido el resultado del entrenamiento de los modelos, en casos muy sencillos y obvios. Para eliminar esta sensación sería bueno probar este desarrollo con nuevos *datasets* que nos dieran otra visión de resultados. Del mismo modo, una variación en los parámetros aplicados podría ayudar en este sentido, aunque fuera a costa de obtener peores resultados en la precisión.

Nos hubiera gustado poder trabajar con texto en castellano, pero, todo y que el interés por idiomas distintos del inglés va en aumento, aún existe una falta de recursos relacionados con otros idiomas. Entre otras cosas la falta de *datasets* disponibles y con un mínimo de información para el entrenamiento nos imposibilita un estudio de este tipo. Sin embargo, la fuente de léxico más común utilizada es WordNet, y ya existe en otros idiomas además del inglés, por lo que es cuestión de tiempo el poder abordar tareas utilizando el idioma propio.

6. Glosario

- ASCII: *American Standard Code for Information Interchange*
- *AWD-LSTM: ASGD Weight-Dropped LSTM*
- BoW: *Bag-of-Words*
- CBoW: *Common Bag of Words*
- CNN: *Convolutional Neural Network*
- CPU: *Central Processing Unit*
- CRISP-DM: *Cross Industry Standard Process for Data Mining*
- CSS: *Cascading Style Sheets*
- CSV: *Comma-Separated values*
- FN: Falso Negativo
- FP: Falso Positivo
- GPU: *Graphics Processing Unit*
- HTML: *HyperText Markup Language*
- IA: Inteligencia Artificial
- IMDB: *Internet Movie Database*
- JS: *JavaScript*
- K-NN: *K-Nearest Neighbor*
- LR: *Logistic Regression*
- MNB: *Multinomial Naïve Bayes*
- ML*: Machine Learning*
- NLP: *Natural Language Processing*
- PCA*: Principal Component Analysis*
- PHP: *Hypertext Preprocessor*
- PwC: *PriceWaterhouseCoopers*
- RAM: *Random Access Memory*
- SSD: *Solid-State Disk*
- SO: Sistema Operativo
- SVM: *Support Vector Machines*
- Tf-idf: *Term Frequency – Inverse Document Frequency*
- TFG: Trabajo Fin de Grado
- TPU: *Tensor Processing Unit*
- ULMFiT: *Universal Language Model Fine-Tuning*
- VN: Verdadero Negativo
- VP: Verdadero Positivo

7. Bibliografía

[1] PwC, «US Edition:Entertainment & Media Outlook 2020-2024,» 2020. [En línea]. Available: https://www.pwc.com/us/en/industries/tmt/library/global-entertainment-media-outlook.html. [Último acceso: 10 Feb 2021].

[2] S. Kumar, «The Beginning of Natural Language Processing,» 4 Dic 2019. [En línea]. Available: https://towardsdatascience.com/the-beginning-of-natural-language-processing-74cce2545676. [Último acceso: 11 Feb 2021].

[3] W. Vorhies, «CRISP-DM – a Standard Methodology to Ensure a Good Outcome,» 26 Jul 2016. [En línea]. Available: https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome. [Último acceso: 11 Feb 2021].

[4] B. Liu y L. Zhang, «A Survey of Opinion and Sentiment Analysis,» de *Mining Text Data*, Chicago, Springer Science+Business Media, 2012, p. 415.

[5] B. Liu, «Problem Definitions,» de *Sentiment Analysis and Opinion Mining*, Chicago, Morgan & Claypool Publishers, 2012, p. 19.

[6] N. Jindal y B. Liu, «Identifying comparative sentences in text documents,» de *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, New York, ACM Press, 2006, pp. 244-251.

[7] B. Liu, «Subjectivity and Emotion,» de *Sentiment Analysis and Opinion Mining*, Chicago, Morgan & Claypool Publishers, 2012, pp. 27-28.

[8] B. Liu, «Affect, Emotion and Mood,» de *Sentiment Analysis - Mining Options, Sentiments and Emotions*, Cambridge, Cambridge University Press, 2020, p. 40.

[9] A. Yessenalina, Y. Yue y C. Cardie, «Multi-level Structured Models for Document-level Sentiment Classification,» de *Proceedings of the 2010 Conference onEmpirical Methods in Natural Language Processing* , Massachusetts, Association for Computational Linguistic, 2010, pp. 1046-1056.

[10] T. Wilson, P. Hoffman, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff y S. Patwardhan, «OpinionFinder: A System for Subjectivity Analysis,» de *Proceedings of HLT/EMNLPon Interactive Demonstrations, Association for Computational Linguistics,*, Vancouver, Association for Computational Linguistics, 2005, p. 34–35.

[11] T. Thura Thet, J.-C. Na y C. S.G. Khoo, «Aspect-based sentiment analysis of movie reviews on discussion boards,» *Journal of Information Science,* 2010.

[12] S. Xie y Y. Liu, «Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,» de *Using corpus and knowledge-based similarity measure in Maximum Marginal Relevance for meeting summarization*, Las Vegas, 2008, pp. 4985-4988.

[13] N. Jindal y B. Liu, «Proceedings of the 16th international conference on

World Wide Web,» de *Review spam detection*, 2007, pp. 1189-1190.

[14] E. Filatova, «roceedings of the Eighth International Conference onLanguage Resources and Evaluation,» de *Irony and sarcasm: corpus generation and analysis using crowdsourcing*, Estambul, 2012, pp. 392-398.

[15] W. Medhat, A. Hassan y H. Korashy, «Sentiment analysis algorithms and applications: A survey,» *Ain Shams Engineering Journal,* 2014.

[16] D. L. Yse, «Your Guide to Natural Language Processing (NLP),» 15 Ene 2019. [En línea]. Available: https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1. [Último acceso: 25 03 2021].

[17] M. Mayo, «A General Approach to Preprocessing Text Data,» KDNuggets, 01 Dic 2017. [En línea]. Available: https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html. [Último acceso: 25 03 2021].

[18] E. Cambria, S. Poria, A. Gelbukh y M. Thelwall, «Sentiment Analysis Is a Big Suitcase,» *IEEE Intelligent Systems,* vol. 32, pp. 74-80, 2017.

[19] A. Veilumuthu, A. Krishnamurthy, C. Veni Madhavan y V. Suresh, «A non-syntactic approach for text sentiment classification with stopwords,» de *Proceedings of the 20th International Conference on World Wide Web*, Hyderabad,, 2011.

[20] J. Brownlee, «A Gentle Introduction to the Bag-of-Words Model,» Machine Learning Mastery, 7 Ago 2019. [En línea]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/. [Último acceso: 25 03 2021].

[21] Z. Li, «A Beginner's Guide to Word Embedding with Gensim Word2Vec Model,» Towards Data Science, 30 May 2019. [En línea]. Available: https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92. [Último acceso: 25 Mar 2021].

[22] I. Guyon y A. Elisseeff, «An Introduction to Variable and Feature Selection,» de *Journal of Machine Learning Research*, MIT Press Cambridge, 2003, pp. 1157-1182.

[23] J. Cai, J. Luo, S. Wang y S. Yang, «Feature selection in machine learning:anew perspective,» de *Neurocomputing*, Changsha, Elsevier, 2018, pp. 70-79.

[24] A. Jovic, K. Brkic y N. Bogunovic, «A review of feature selection methods with applications,» de *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, 2015.

[25] C. Musto, G. Semeraro y M. Polignano, «A comparison of Lexicon-based approachesfor Sentiment Analysis of microblog posts,» de *Proceedings of the 8th International Workshop on Information Filtering and Retrieval*, Bari, 2014.

[26] V. Hatzivassiloglo y K. R. McKeown, «Predicting the semantic orientation of adjectives,» *ACL '98/EACL '98: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics,* pp.

174-181, 1998.

[27] B. Liu, «Unsupervised Learning,» de *Web Data Mining*, Chicago, Springer, 2008, pp. 117-149.

[28] B. Liu, «Supervised Learning,» de *Web Data Mining*, Chicago, Springer, 2008, pp. 55-116.

[29] J. Le, «A Gentle Introduction to Neural Networks for Machine Learning,» Codementor Comunity, 14 Sep 2018. [En línea]. Available: https://www.codementor.io/@james_aka_yale/a-gentle-introduction-to-neural-networks-for-machine-learning-hkijvz7lp. [Último acceso: 29 Mar 2021].

[30] I. Rish, «An empirical study of the naive Bayes classifier,» *IJCAI 2001 workshop on empirical methods in artificial intelligence,* vol. 3, nº 22, pp. 41-46, 2001.

[31] J. Avila Camacho, «Clasificador Naive Bayes,» JacobSoft, 4 May 2020. [En línea]. Available: https://www.jacobsoft.com.mx/es_mx/clasificador-naive-bayes/. [Último acceso: 29 Mar 2021].

[32] C. Molnar, «Logistic Regression,» de *Interpretable Machine Learning*, Munich, Leanpub, 2020, pp. 71-79.

[33] J. Lin y A. Kolcz, «Large-Scale Machine Learning at Twitter,» *SIGMOD '12: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data,* pp. 793-804, 2012.

[34] S. Faltl, M. Schimpke y C. Hackob, «ULMFiT: State-of-the-Art in Text Analysis,» Seminar Information Systems (WS18/19), 7 Feb 2019. [En línea]. Available: https://humboldt-wi.github.io/blog/research/information_systems_1819/group4_ulmfit/. [Último acceso: 8 Abr 2021].

[35] H. Kriplani, «Understanding language modelling(NLP) and Using ULMFIT,» Towards Data Science, 4 Sep 2019. [En línea]. Available: https://towardsdatascience.com/understanding-language-modelling-nlp-part-1-ulmfit-b557a63a672b. [Último acceso: 8 Abr 2021].

[36] M. Kantrowitz, B. Mohit y V. Mittal, «Stemming and its effects on TFIDF Ranking (poster session),» 01 Jul 2000. [En línea]. Available: http://www.kramirez.net/RI/Material/Internet/sigir2000.pdf. [Último acceso: 14 May 2021].

[37] K. Shah, H. Patel, D. Sanghvi y M. Shah, «A Comparative Analysis of Logistic Regression, Random Forest and KNN Models for the Text Classification,» Springer, 5 Mar 2020. [En línea]. Available: https://link.springer.com/article/10.1007/s41133-020-00032-0. [Último acceso: 10 Abr 2021].

[38] M. Porter, «The Porter Stemming Algorithm,» -, 01 Ene 2006. [En línea]. Available: https://tartarus.org/martin/PorterStemmer/. [Último acceso: 12 Abr 2021].

[39] L. N. Smith, «A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay,» US Naval Research Laboratory Technical Report, Washington, 2018.

8. Anexos

En este documento no se adjuntarán los ficheros fuente de código ni cualquier otro utilizado para el desarrollo, para no incrementar el tamaño. Con el fin de compartir toda la información y poder testear el trabajo se ha habilitado un repositorio GitHub[10], accesible de forma pública en el enlace: https://github.com/jchulilla/TFG.

A modo resumen, el contenido del repositorio se estructura en cuatro partes:

- *Back-End* Python
    - *"TFG_jchulilla_PEC3.py"*. Código Python que entrena, crea y prueba los modelos NLP correspondientes a los algoritmos LR y MNB. Generado y probado con la herramienta *Jupyter Notebook 6.1.4*. La versión de software Python utilizada es la 3.8.5.
    - *"TFG_jchulilla_PEC3(ulmfit).py"*. Código Python que entrena, crea y prueba el modelo NLP correspondiente al algoritmo ULMFiT. En este caso se ha utilizado la solución web "*Google Colab*" que en la fecha de las pruebas utilizaba la versión 3.7.10 de Python.
    - Ficheros pkl. Son los ficheros que contienen los modelos generados en los puntos anteriores y el vectorizador. La parte *front-end* sólo tendrá que cargar estos modelos y utilizarlos, no es necesario volver a entrenar. El fichero *"learn.pkl"* correspondiente al modelo *fastai* para ULMFiT, por su tamaño, se ha comprimido y partido en 4 ficheros con nomenclatura *"learn.zip.001-004".*
- *Front-End* web
    - *"PhpProjectTest"*. Carpeta que contiene el código HTML, CSS y PHP estructurado por carpetas. La parte de desarrollo se ha realizado utilizando la herramienta *Apache NetBeans IDE 12.0*, por lo que la estructura de carpetas corresponde a un proyecto realizado con este software. Para las pruebas se ha utilizado *XAMPP Control Panel v3.2.4*, que nos ha permitido crear un servidor Apache en local.
- *Front-End* Python
    - *"TFG_jchulilla_PEC3_TEST_xxx.py"*. Son 3 ficheros Python, uno por cada modelo probado en este TFG. Son los encargados de llamar a los modelos previamente creados en los ficheros *back-end* de Python. Se han desarrollado y probado con la misma infraestructura *Jupyter Notebook* y Python utilizada para el fichero *"TFG_jchulilla_PEC3.py"* comentada en el primer punto.

[10] https://github.com/jchulilla/TFG

- *Deploy*

    - *“Dockerfile”*. Fichero de configuración utilizado para la generación de la imagen Docker que permite la ejecución del *front-end web*. Basada en una imagen Docker PHP 7.2-apache, añade la instalación de la última versión Python disponible.

    - *“Requirements.txt”*. Este fichero indica a “Dockerfile” los requerimientos de software necesarios. En nuestro caso las versiones de las librerías que nos interesan para los *Front-end* Python:
        - sklearn==0.0
        - pickleshare==0.7.5
        - fastai==1.0.61
        - torch==1.8.1
        - spacy==2.2.4

README.md

TFG

TFG Informatica UOC 2021

PhpProjectTest --> Codigo HTML/CSS/PHP que incluye la parte front-end
TFG_jchulilla_PEC3(ulmfit).py --> Codigo Python con la generacion del modelo ULMFiT
TFG_jchulilla_PEC3.py --> Codigo Python con la generacion de los modelos LR y MNB
TFG_jchulilla_PEC3_TEST_LR.py --> Codigo Python con la llamada al modelo LR
TFG_jchulilla_PEC3_TEST_MNB.py --> Codigo Python con la llamada al modelo MNB
TFG_jchulilla_PEC3_TEST_ULM.py --> Codigo Python con la llamada al modelo ULMFiT

Dockerfile --> Generacion del docker
requirements.txt --> Librerias necesias python para el Docker

learn.pkl --> Modelo python ULMFiT (comprimido por volumenes en zip)
modelLR.pkl --> Modelo python LR
modelMNB.pkl --> Modelo python MNB
tfidf_vectorizer.pkl --> Vectorizador python

Figura 34. Fichero README.MD