ANÁLISIS DE SENTIMIENTO EN TIEMPO REAL DE MENSAJES DE TWITCH CON ALGORITMOS DE CLASIFICACIÓN

RAÚL SÁENZ RUBIA
GRADO EN INGENIERÍA INFORMÁTICA
INTELIGENCIA ARTIFICIAL

CONSULTOR: FERRAN DIEGO ANDILLA
Profesor/a responsable de la asignatura: CARLES VENTURA ROYO

Fecha de entrega: Enero de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis de sentimiento en tiempo real de mensajes de Twitch con algoritmos de clasificación*
Nombre del autor:	*Raúl Sáenz Rubia*
Nombre del consultor/a:	*Ferran Diego Andilla*
Nombre del PRA:	Carles Ventura Royo
Fecha de entrega (mm/aaaa):	*01/2020*
Titulación o programa:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Español*
Palabras clave:	*Análisis sentimental, Procesamiento de Lenguaje Natural, aprendizaje computacional*

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

El análisis de sentimiento es un subcampo del Procesamiento del Lenguaje Natural cuyo objetivo es analizar textos subjetivos y extraer información de utilidad sobre la polaridad de sentimiento del autor a nivel de documento, de frase concreta o sobre el aspecto de un artículo o servicio de interés para el analista.

En concreto, las redes sociales y páginas de micro-blogging como Twitter o Twitch se han convertido en fuentes inagotables de textos que pueden ser usados para analizar la opinión del público sobre aspectos tan variados como la política, economía o cualquier otro producto o servicio al instante.

Twitch, en concreto, es una plataforma que ha crecido enormemente durante los últimos años, lo cual hace necesario mejorar la interacción con el público para desmarcarse de la competencia, y el chat de Twitch es una fuente de información que puede ayudar a los creadores de contenido a conocer la opinión de su audiencia. Sin embargo, actualmente no existen extensiones para el análisis de sentimiento en español para dicha plataforma, por lo que vemos una oportunidad para innovar.

En este trabajo se utilizan métodos de aprendizaje profundo MLP, CNN y LSTM para la construcción de un clasificador de polaridad de sentimiento con tres niveles: positivo, neutro y negativo, con el que se obtienen resultados del 76% de exactitud media, lo cual es bastante aceptable teniendo en cuenta los obstáculos hallados en el análisis como la ambigüedad del lenguaje, el desconocimiento del contexto y las errores ortográficos, intencionados o no.

Abstract (in English, 250 words or less):

Sentiment analysis is a subfield of Natural Language Processing whose objective is to analyze subjective texts and extract useful information about the author's polarity of feeling at the document level, of a specific phrase or about an aspect of an product or service of interest for the analyst.

Specifically, social networks and micro-blogging pages such as Twitter or Twitch have become inexhaustible sources of texts that can be used to instaltly analyze public opinion on aspects as varied as politics, economy or any other product or service.

Twitch, in particular, is a streaming platform that has grown tremendously in recent years, which makes it necessary to improve interaction with the public to get rid of the competition, and Twitch chat is a source of information that can help creators of content to know the opinion of your audience. However, there are currently no extensions for sentiment analysis in Spanish for this platform, so we see an opportunity to innovate.

In this work, we use MLP, CNN and LSTM deep learning methods to develop a multi-class sentiment analysis classifier with three levels of polarity: positive, neutral and negative, with achieved average results of 75% in accuracy, which is a quite acceptable outcome considering the obstacles found in the analysis such as language ambiguity, unawareness of context and intentional or unintentional misspellings.

Índice

Índice de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Twitch es una plataforma de *streaming* propiedad de Amazon desde 2014 con sede en San Francisco que, desde su presentación el 6 de juny de 2011, se ha consolidado como una alternativa a Youtube bastante atractiva para la retransmisión de contenido de video en directo, especialmente sobre videojuegos.

El crecimiento de la plataforma ha sido enorme si tenemos en cuenta las estadísticas oficiales[1]: ha pasado de tener 300 mil emisores únicos el año 2012 a tener 3.4 millones, lo cual tiene un efecto negativo en el número de espectadores por canal al incrementarse la oferta de contenido.

Con una competencia tan grande es imprescindible no sólo ofrecer contenido atractivo para tu audiencia sino también cuidar de otros aspectos del canal y obtener *feedback* sobre tus retransmisiones en tiempo real como la polaridad de sentimiento.

Desde el punto de vista de un desarrollador de contenidos, es muy útil disponer de herramientas que faciliten la gestión de la satisfacción de los espectadores y tener un resumen visual del sentimiento del chat en tiempo real. Twitch proporciona un sistema de extensiones que permite extender las funcionalidades de la plataforma por medio de paneles o sobre impresiones interactivas. No obstante, en la actualidad las pocas extensiones disponibles para el análisis de sentimiento son muy básicas con respecto a la información proporcionada y sólo funcionan en idioma inglés, por lo que no hay ninguna alternativa para aquellos usuarios que quieran analizar el sentimiento del chat en español o catalán.

Es por este motivo que, con este proyecto, queremos llenar el vacío presente en este mercado mediante el desarrollo de una extensión para esta plataforma que implemente un clasificador para el análisis en tiempo real del sentimiento del chat utilizando técnicas de *machine learning* y más concretamente modelos de aprendizaje profundo.

1.2 Objetivos del Trabajo

1.2.1 Objetivo principal:

El objetivo principal del proyecto es la creación de un clasificador multiclase para la clasificación de textos según su polaridad de sentimiento en las categorías positivo, negativo y neutro y la posterior integración del modelo en una extensión de Twitch para llevar a cabo el análisis de sentimiento de los mensajes en español del chat de esta plataforma para proporcionar a creadores

1 Font: https://twitchtracker.com/statistics

de contenido una herramienta para mostrar de forma visual información resumida sobre la polaridad del chat mediante una interfaz gráfica de usuario.

1.2.2 Objetivos específicos:

- Análisis del estado del arte del análisis de sentimiento en idioma español.
- Análisis y aplicación de métodos de limpieza y preprocesamiento de textos a los mensajes del corpus.
- Análisis y comparación de rendimiento de diversos algoritmos de aprendizaje profundo como MLP, redes convolucionales y LSTM. mediante métrica *Accuracy*.
- Estudio de la arquitectura de una extensión para Twitch.
- Desarrollo de la extensión integrando el clasificador elegido, la cual se divide en la creación del front-end (GUI) en HTML / Javascript y la extensión back-end service (EBS).

1.3 Enfoque y método seguido

El proyecto se divide en dos fases principales: por un lado el desarrollo del clasificador para el análisis de sentimiento y por otra parte la extensión de Twitch que mostrará el resultado de este análisis.

En cuanto al desarrollo del clasificador, se estudiará el estado del arte del análisis de sentimiento mediante diversas técnicas de clasificación de aprendizaje automático como Naive Bayes, Árboles de Decisión, Regresión Logística o SVM y se comparará el rendimiento obtenido con técnicas de aprendizaje profundo como son las redes neuronales multicapa, redes convolucionales o las redes neuronales recurrentes.

Durante la ejecución de esta fase, seguiremos el ciclo de trabajo estándar de un proyecto de procesado de lenguaje natural como es el CRISP-DM (figura 1). Las fases principales son las siguientes:

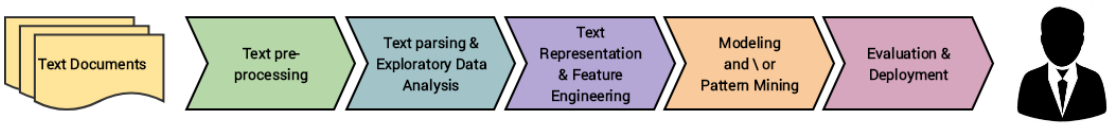


Figura 1: Flujo de trabajo de un proyecto de PLN. Fuente:[1]

A continuación se detalla cada un de los pasos de la metodología:

- Se partirá de un corpus de documentos de texto etiquetados con su polaridad de sentimiento: positivo, negativo o neutro (o con granularidad más fina).

- Preprocesamiento de los textos:
  - Traducción de los textos al castellano, si es necesario.
  - Conversión de los textos a minúsculas.

- Eliminación de palabras vacías sin significado semántico (*stop words*).
- Eliminación de caracteres especiales.
- Conversión de los textos a vectores de palabras (tokenización).
- Derivación (*stemming*) / Lematización.

- Obtención de los vectores de características mediante la transformación de las palabras en vectores numéricos.

- Entrenamiento y evaluación de los modelos de clasificación. Dividiremos el conjunto de datos en un conjunto de entrenamiento que contendrá el 80% de los ejemplos y dejaremos el 20% restante para la evaluación de los modelos utilizando validación cruzada.

Fase 2: Desarrollo de la extensión de Twitch

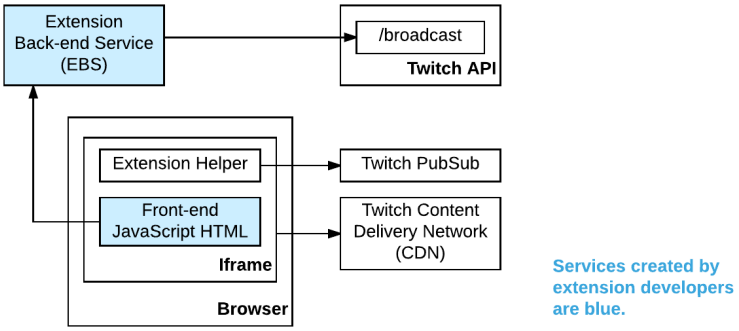


Figura 2: Estructura de'una extensió de Twitch. Font:[2]

En la figura figura 2 podemos ver que dos de los componentes de la extensión que hay que crear son la *Extension Back-end Service* (EBS), que se encarga de la parte de la lógica del dominio, y un front-end en JavaScript y HTML encargada de visualizar la respuesta del EBS.

Para el desarrollo de la extensión se partirá de un ejemplo existente y se modificará para adaptarlo a la funcionalidad deseada. De esta forma el desarrollo será más ágil dado que el tiempo disponible es bastante ajustado.

1.4 Planificación del Trabajo

A continuación se exponen la Estructura de Descomposición del Proyecto, la planificación temporal (Diagrama de Gantt) y el Presupuesto económico para la ejecución del proyecto.

1.4.1 EDP

La EDP (figura 3) descompone jerárquicamente el trabajo definido por el alcance del proyecto en seis fases principales que coinciden con los seis entregas

de evaluación continua del Trabajo de Fin de Grado. Cada fase se desglosa en tareas y subtareas más específicas que se deben completar antes de pasar al nivel siguiente y cada subnivel del nivel principal termina con la entrega de una parte del proyecto.

1.4.2 Planificación temporal (Diagrama de Gantt)

La planificación (figura 4) comprende el periodo del 09/18/19 hasta el 01/09/20 e incluye desde la propuesta inicial del proyecto hasta la redacción de la memoria y la elaboración de la presentación.

El plazo total para la realización del proyecto serán 114 días naturales de lunes a domingo (festivos incluidos) con una dedicación aproximada de 300 horas. La dedicación diaria será de 2.63 horas de media, que resultan de dividir las 300 horas entre el total de 114 días.

Figura 3: EDP del proyecto. Fuente: elaboración propia

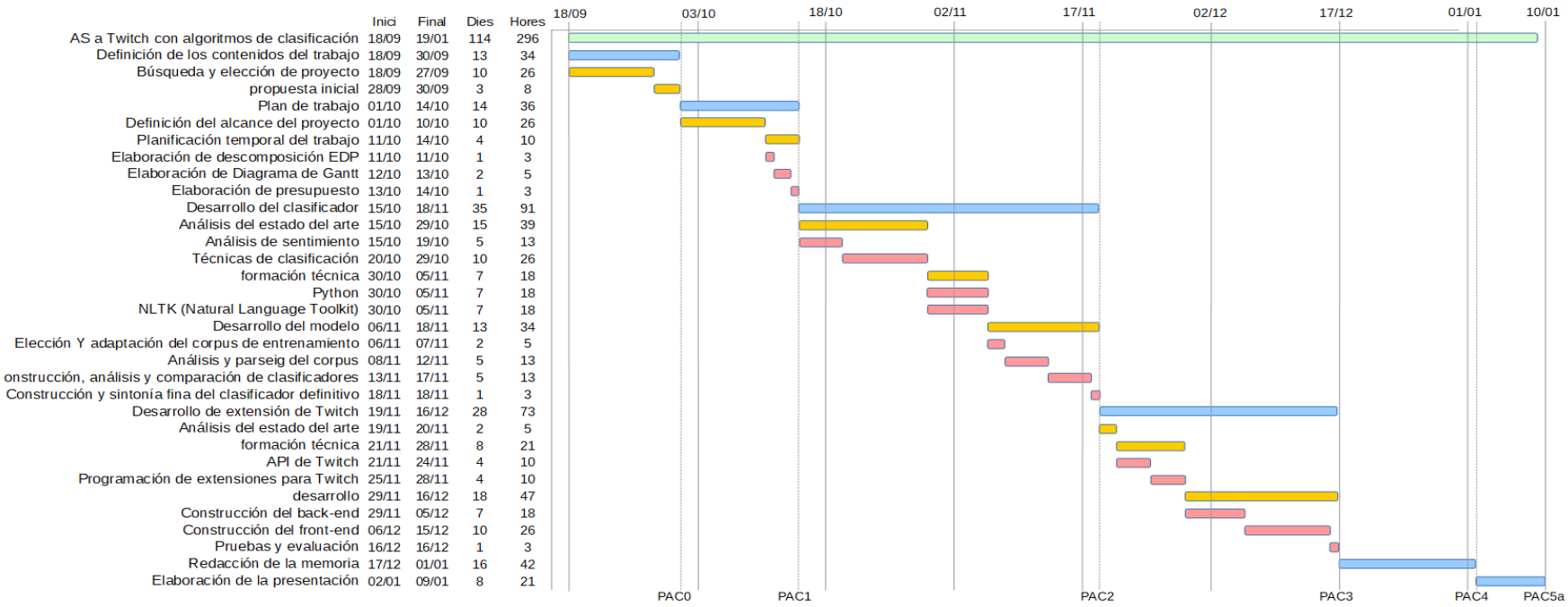


Figura 4: Diagrama de Gantt de la planificación inicial

1.4.3 Presupuesto

Para la realización de este trabajo se ha utilizado *hardware* existente y *software* abierto. Sin embargo, se calcula el coste derivado de este proyecto si éste se hubiera producido en el contexto de una empresa o como trabajador autónomo.

Lo primero que tenemos en cuenta es el importe de matriculación de los 14 ECTS del Trabajo Final de Grado: 564,84 €.

A continuación calculamos el coste derivado de los recursos humanos involucrados en el proyecto (excluidos tutor y profesores), en este caso es sólo el salario del estudiante. Consideramos el salario anual medio de un ingeniero informático de 26.654 euros brutos, divididos entre los 248 laborables del año 2019 y multiplicados por 8 horas / día que son 1.984 horas. Así pues, el salario por hora resultante es de 13,43 € / hora que, multiplicados por las 296 horas del proyecto resultan en 3.976,60 €.

En cuanto a los equipos informáticos y software, se utiliza hardware existente consistente en un ordenador de sobremesa valorado en unos 1000 € aproximadamente. Teniendo en cuenta que este equipo se renueva cada 4 años, se han de amortizar 250 euros / año. Así pues, 250 euros entre 1984 horas (284 laborables x 8 horas) = 0.126 € / hora x 296 h = 37,29 €

Finalmente, tenemos en cuenta el consumo eléctrico derivado de la realización del proyecto consistente en el consumo del ordenador y pantalla con una potencia total de 1000 Watts a un precio de kWh de 0,1350:

Tabla 1: Deglose económico del proyecto

Concepto	**Unidades**	**Precio**	**Importe**
Matrícula TFG	1	564,84	564,84
Sueldo ingeniero informático	296	13,43	3976,60
Amortización hardware	296	0,126	37,29
Consumo eléctrico	296	0,1350	39,96
TOTAL			**4618,69**

1.4.4 Análisis de riesgos

Tabla 2: Relación de riesgos en la ejecución del proyecto.

Riesgo	**Posible solución**	**Solución adoptada**	**Impacto**
Licencia del entorno de desarrollo caducada.	Renovar la licencia o usar un IDE de Python gratuito como Spyder.	Se renueva la licencia del entorno PyCharm de forma gratuita por ser estudiante de la UOC.	Bajo
Retraso en el inicio de formación técnica según la planificación.	Llevar a cabo la formación técnica paralelamente al desarrollo del modelo.	Llevar a cabo la formación técnica paralelamente al desarrollo del modelo.	Bajo
No se dispone de un corpus etiquetado para el análisis de sentimiento en español.	Usar un corpus en inglés y traducirlo al español.	-	Alto
Se obtiene un corpus en español, pero pertenece a un dominio diferente al del problema que queremos resolver.	No es recomendable usar un corpus de otro dominio. Buscar un corpus adecuado para el dominio en otro idioma y traducirlo.	-	Alto
No existen paquetes de PLN en español	Desarrollar el modelo en inglés.	Se utiliza NLTK para obtener las palabras vacías y Spacy para la lematización.	Medio/Alto
Tras entrenar varios modelos no se consigue la precisión esperada.	Posiblemente el corpus sea pequeño o poco variado. Analizar los resultados devueltos y la matriz de confusión. Si se predicen muchos casos de una clase incorrectamente, ampliar el número de documentos etiquetados de esa clase.	-	Medio
No se dispone de *hardware* adecuado para el entrenamiento de redes neuronales complejas.	Contratar un servicio de computación en la nube como Google Cloud desde $0,29/hora de proceso de GPU Nvidia Tesla T4 con 16 GB de GDDR6.	-	Medio

1.5 Breve resumen de productos obtenidos

Una vez finalizado el proyecto se habrá obtenido un modelo de clasificación multiclase para analizar en tiempo real el sentimiento de los mensajes del chat de Twitch y clasificarlos con tres niveles de polaridad: positiva, negativa o neutra. Se entregará la memoria del proyecto especificando todo el proceso de desarrollo de dicho modelo, las decisiones tomadas y los problemas encontrados, así como un archivo comprimido con el código fuente utilizado para procesar y analizar el corpus de mensajes, crear el vocabulario, preparar los datos para su utilización en el modelo e implementar los diferentes modelos de clasificación probados.

Adicionalmente, se incluirá los archivos creados durante el desarrollo de la extensión de Twitch tanto del *frontend* como del EBS para su libre uso y adaptación a otros proyectos según indica la licencia de reconocimiento 3.0 de Creative Commons.

1.6 Breve descripción de los otros capítulos de la memoria

En el capítulo 2 de la memoria de este Trabajo Final de Grado se tratará el Estado del Arte del procesamiento de lenguaje natural poniendo especial énfasis en el análisis de sentimiento o la minería de opinión y veremos qué problemáticas surgen, las soluciones aportadas y los retos actuales del AS, como el sarcasmo y la ironía, las diferentes formas de negación, la ambigüedad del lenguaje y la polaridad múltiple. En la segunda parte de este capítulo se hará un repaso de los algoritmos de clasificación más utilizados para esta tarea y cuáles son las ventajas y desventajas de cada uno de ellos.

En el capítulo 3 se detallará la metodología utilizada durante el proyecto para la elaboración y evaluación de los modelos de clasificación, dedicando un apartado para cada una de las arquitecturas siguientes: MLP, LSTM y CNN y mostrando los resultados obtenidos.

Los capítulos 4 y 5 se dedican a la implementación de la extensión de Twitch. El capítulo 3 hará un breve estudio de las extensiones y aplicaciones similares que existen actualmente en el mercado, En el capítulo 4 se explica el proceso de implementación de la extensión en sí misma.

El último capítulo lo reservamos para las conclusiones, lecciones aprendidas y futuras mejoras del producto.

2. Análisis del estado del arte

2.1 Introducción

Aunque actualmente consideramos el análisis de sentimiento (AS) como una actividad relativamente reciente llevada a cabo con computadores, lo cierto es que ha sido una actividad que el hombre ha llevado a cabo desde la antigüedad para la ayuda en la toma de decisiones de uno u otro tipo.

A pesar de la voluntad de la comunidad científica por llevar a cabo investigaciones, durante las décadas siguientes no se realizan muchos avances en este campo debido a la escasez de textos opinados a su alcance.

No es hasta la llegada de Internet y la apertura al público en 1991 de la *World Wide Web* cuando los investigadores ganan acceso a extensas fuentes de opinión, gracias al contenido generado por los usuarios y a multitud de conjuntos de datos etiquetados por terceros. Hasta ese momento, las únicas herramientas de las que disponían las empresas o instituciones para recopilar información eran encuestas, formularios, grupos de estudio, entro otros métodos tradicionales de sondeo.

El incremento exponencial de contenido opinado en la red en foros de opinión, páginas de micro *blogging* y de valoración de artículos, servicios, etc. generó un estallido de popularidad del análisis de sentimiento y el interés de los investigadores por el desarrollo de métodos para la minería automática de esta enorme cantidad de información de forma más rápida y eficiente [2,3].

Paralelamente, el aumento de la literatura científica relacionada con el análisis de sentimiento crece como la espuma. (Mäntylä, Mika V., et al., 2017) recogen en "The Evolution of Sentiment Analysis—A Review of Research Topics, Venues, and Top Cited Papers." el gran incremento en el número de estudios realizados sobre el tema desde el año 2004, especialmente centrados en las opiniones de artículos en la Web, que desde entonces se ha extendido a otras áreas como el análisis de los mercados financieros y las reacciones tras ataques terroristas, aunque sus áreas de aplicación práctica son todas aquellas susceptibles de generar opinión de interés.

En la actualidad (2010-Act.), la existencia de *hardware* (especialmente procesadores gráficos o GPUs) de gran potencia computacional y la computación en la nube hacen posible el entrenamiento de modelos para el análisis de sentimiento mucho más sofisticados como los modelos de aprendizaje profundo, dejando una puerta abierta para la mejora.

2.1.2 Definición

(Mäntylä, Mika V., et al, 2017) define el análisis de sentimiento de la siguiente manera:

> *"Sentiment analysis is a series of methods, techniques, and tools about detecting and extracting subjective information, such as opinion and attitudes, from language"* [El análisis de sentimiento es un conjunto de técnicas y herramientas para la detección y extracción de información subjetiva, tales como opinión y actitudes, del lenguaje].

(Catardo Musto et al, 2014) por su parte afirma:

> *"A recent trend is to analyze people feelings, opinions and orientation about facts and brands: this is done by exploiting Sentiment Analysis techniques, whose goal is to classify the polarity of a piece of text according to the opinion of the writer"* [Una tendencia reciente es la de analizar los sentimientos, opiniones y otrientación de la gente acerca hechos y marcas: esto se hace utilizando técnicas de análisis de sentimientos, la meta de los cuales es clasificar la polaridad de un fragmento de texto de acuerdo con la opinión del autor].

Así pues, se puede definir el análisis de sentimiento como un tipo de clasificación de textos que consiste, por una parte, en detectar y extraer información relativa a la opinión del autor de un texto acerca de una determinada entidad y, por otra parte, trata de clasificar dicho texto de acuerdo con dicha polaridad.

2.2. Clasificación de análisis de sentimiento según el alcance del análisis

El análisis de sentimiento se puede realizar a diferentes niveles, dependiendo del objetivo que se persiga [7][28]:

a. Análisis a nivel de documento: el análisis trata de extraer información sobre la opinión general de un documento formado por varias frases, cada una de ellas con su propia polaridad.

Para tratar esta problemática existen enfoques supervisados y no supervisados. Los supervisados consisten en aplicar técnicas de *machine learning* como SVM, kNN, Naive Bayes o Regresión Logística entrenando el modelo con un corpus previamente etiquetado con un conjunto de clases bipolares (negativo, positivo) y opcionalmente añadiendo una clase extra: la clase neutra.

El método no supervisado consiste en determinar la orientación semántica (OS) de varias frases dentro del documento. Si la media de OS de dichas frases supera un umbral determinado se clasifican como positivas y en caso contrario como negativas. Para calcular la OS de una palabra o frase (Turney, P., 2002) [29] propone el algoritmo PMI-IR (*Pointwise Mutial Information-Information Retrieval*) entre la frase o palabra y dos palabras de referencia, una para cada categoría: '*excellent*' para la clase positiva y '*poor*' para la negativa. Veremos brevemente este algoritmo en la sección 'Algoritmos de clasificación no supervisados'.

La selección de las frases usando técnicas no supervisadas se realiza mediante patrones POS (*Parts Of Speech*) [6] o bien usando un diccionario de palabras y frases con sentimiento.

b. Análisis a nivel de frase: el análisis trata de identificar y extraer el sentimiento sobre una o varias entidades existente en una frase individual de un documento. Para ello, debemos asegurarnos de dos cosas:

1. La frase contiene únicamente una opinión.
2. La frase es subjetiva y no objetiva.

Para la solución de estos problemas se proponen métodos supervisados como no supervisados, siendo los supervisados los mismos que a nivel de documento. En cuanto al enfoque no supervisado, el enfoque propuesto es similar al propuesto por (Turney, P., 2002) pero usando *log-likelihood ratio* en lugar de PMI para hallar la OS [29].

En esta linea de investigación, (Ellen Riloff, Janyce Wieve, 2003) elabora un proceso de *bootstraping* para aprender patrones de extracción de expresiones subjetivas. El proceso consiste en utilizar un Clasificador de Subjetividad de Alta Precisión sobre un conjunto de textos no etiquetados para hallar frases subjetivas. A partir de este conjunto de frases se utiliza un algoritmo de extracción de patrones para obtener nuevos patrones sintácticos asociados a la subjetividad y finalmente los patrones aprendidos se utilizan para mejorar el rendimiento del clasificador de subjetividad, que puede utilizar dichos patrones para hallar nuevas frases subjetivas [30].

c. Análisis a nivel de aspecto o característica: el análisis trata de identificar y extraer el sentimiento asociado a varias características de un objeto, como por ejemplo el precio, duración de batería o cámara de un teléfono móvil, dentro de un documento o frase.

Dentro de este tipo de análisis debemos distinguir entre dos tipos de aspectos: explícitos e implícitos.

Los primeros son aquellos que aparecen en el documento como frases sustantivales (*noun phrases* en inglés), mientras que los segundos no aparecen explícitamente sino que se infieren a partir de los adjetivos como ‘pesado’ o ‘grande’, que implícitamente hacen referencia al peso y al tamaño respectivamente.

2.3 Preprocesamiento y normalización de textos para la clasificación

a, Preprocesamiento

Una de las tareas previas al entrenamiento de los modelos de clasificación es el preprocesamiento del texto. Generalmente, el corpus de entrenamiento contiene textos sin tratar a los que es necesario aplicar una serie de técnicas de limpieza y

filtrado de características para reducir las dimensión del texto a la mínima necesaria. Las tareas de preprocesamiento más comunes son [14][15][23]:

- **Limpieza de caracteres especiales:** eliminamos cualquier carácter especial que no sea una letra. Lo que obtenemos tras este paso es un texto compuesto única y exclusivamente por palabras separadas por espacios, libre de signos de puntuación y otros caracteres especiales.

- **Eliminación de caracteres repetidos:** en el lenguaje natural escrito encontramos palabras con caracteres repetidos intencionadamente con el objetivo de dar énfasis, como 'muuuy' o 'bieeeen'. Es necesario eliminar dichos caracteres repetidos para obtener el término correcto y evitar así una duplicidad de palabras.

- **Eliminación de stop-words:** otro paso fundamental para reducir la dimensionalidad del espacio de características es eliminar palabras sin significado semántico como artículos, preposiciones y conjunciones. Los sustantivos también podrían ser objeto de eliminación si solamente nos interesase extraer la polaridad del texto y no el aspecto relacionado.

- **Conversión de palabras a minúsculas:** de la misma forma que con la eliminación de caracteres repetidos, conviene transformar todas las palabras a minúsculas para eliminar duplicidades (por ejemplo, entre 'Bien' y 'bien'), siempre y cuando la distinción entre mayúsculas y minúsculas no sea relevante para el objetivo del análisis.

- **Tokenización:** conversión del texto atómico a un vector de items.

b. Normalización

Llamamos normalización al proceso de transformación de un término en una forma canónicas representante de todos sus términos flexionados o derivados para simplificar y reducir el vocabulario reconocido por el modelo. Las técnicas de normalización más utilizadas son el *stemming* y la lematización.

- **Stemming[2]:** consiste en hallar la raíz (en inglés, *stem*) de una palabra. Por ejemplo, tanto 'bibliotecario' como 'biblioteca' comparten la raíz 'biblioteca', por lo que la primera sería reducida a 'biblioteca'. Esta técnica es usada actualmente por el motor de búsqueda de Google para simplificar las cadenas de búsqueda. No obstante, tiene el problema de que se llega a perder el significado semántico de las palabras.

  Los algoritmos más usados en lengua inglesa son los de Porter y Snowball (Wikipedia, 2019).

- **Lematización**: proceso lingüístico que consiste en, dada una palabra flexionada o derivada, hallar su lema o representante canónico. Por ejemplo, el lema de 'produjo' sería 'producir'.

2 https://www.geeksforgeeks.org/introduction-to-stemming/

La lematización se puede llevar a cabo mediante análisis morfológico o sintáctico.

Un estudio realizado por Pu Han et al (2012) [13] demuestra cómo el *stemming* tiene un mayor rendimiento a la hora de reducir la dimensionalidad del espacio de características comparado con la lematización. Una razón es que el *stemming* poda las palabras de acuerdo con su raíz mientras que la lematización está basada en diccionario solamente transforma las palabras en sus correspondientes lemas:

Original	**Porter**	**Snowball**	**Lematización**
general	gener	general	general
generate	gener	general	generate
general	gener	general	general
generic	gener	generic	generic
generous	gener	generous	generous
generating	gener	generat	generating
generously	gener	generous	generously
generated	gener	generat	generated

Tabla 3: Resultados de stemming y lematización. Fuente: [13]

- **Manejo de negaciones:** en el análisis de sentimiento es importante detectar las negaciones dentro de una frase que puedan afectar el significado de las palabras posteriores. Una solución empleada frecuentemente es añadir un prefijo como 'no', 'neg', etc. a todas las palabras posteriores hasta el siguiente punto para diferenciar dichas palabras de sus negaciones [15]

- **Parts of Speech tagging (etiquetado de Partes de la Frase):** esta técnica consiste en utilizar un diccionario para etiquetar las palabras según su función dentro de la frase (sustantivo, verbo, adjetivo, adverbio, etc.).

b. Obtención del vector de características

En cuanto a la obtención del vector de características, se utilizan diversos métodos o técnicas:

Unigramas: en este modelo, cada característica del vector se corresponde con cada una de las palabras contenidas en el corpus de entrenamiento. Así pues, mediante esta técnica, también conocida como *bag of words* (bolsa de palabras, en español), se obtiene un vector disperso de características binarias, donde el

valor 1 indica la presencia del término en el texto de ejemplo y el valor 0 su ausencia. Por ejemplo, si disponemos del vocabulario siguiente:

buena	no	contento	formidable	...	horrible

El vector de características de la frase "**No** me gustó la película" sería:

0	1	0	0	...	0

Mientras que el de "...me pareció una **buena** actuación" sería:

1	0	0	0	...	0

Este modelo sufre de diversos problemas como por ejemplo que no tiene en cuenta el orden de las palabras dentro del documento y las grandes dimensiones de los vectores.

Una alternativa a los vectores **basados en presencia** o **cuenta binaria** es la **basada en la frecuencia** de cada término en el texto, aunque los resultados mostrados por el estudio de (Pang et al.,2002) muestran peor rendimiento tanto de Naive Bayes como de SVM, lo cual entra en contradicción con el estudio realizado por (McCallum and Nigan, 1998), donde tener en cuenta la frecuencia de las características sí tenía una relevancia significativa. No obstante, hay que tener presente que el trabajo de (McCallum and Nigan, 1998) se centra en la clasificación de textos basada en asuntos, por lo que es razonable pensar que la repetición de un término sí es relevante a la hora de clasificar un texto según el tema al que hace referencia, pero no tanto a la hora de extraer el sentimiento asociado al texto.

Por otra parte, es obvio que en textos muy cortos como tuits o microblogs la diferencia con respecto al modo basado en presencia es inapreciable ya que las frecuencias absolutas son prácticamente de 1 o 0 y algunas de las palabras con una frecuencia más alta suelen carecer de significado, como artículos y preposiciones. Para solventar este problema, una solución popular es normalizar las frecuencias con la medida TF-IDF (Term Frequency-Inverse Document Frequency):

$$tf(t,d)=f_{t,d}$$

$$idf(t,D)=\log\frac{N}{|d\in D:t\in d|}$$

$$tfidf(t,d,D)=tf(t,d)\cdot idf(t,D)$$

Donde t es un término, d un documento, D un conjunto de documentos (o corpus) y N el número de documentos del corpus.

Así pues, el peso de una palabra crece cuantas más veces aparece dentro de un documento, pero disminuye en cuantos más documentos aparezca.

Bigramas (y generalizando, N-gramas): consiste en crear características consistentes en conjuntos de N palabras consecutivas. Tiene la ventaja de que incluye cierta información posicional acerca de las palabras que componen el documento. No obstante, (B. Pang, L. Lee, 2002) en su estudio [4] no obtienen un incremento en el rendimiento con el uso de bigramas basados en presencia en ninguno de los tres clasificadores probados (NB, ME, SVM).

Partes del Discurso (POS): Otro de los enfoques propuestos es el de añadir información sobre el tipo de cada palabra en el vector de características puesto que intuitivamente los adjetivos transportan gran cantidad de información sentimental que otras palabras como sustantivos o verbos.

No obstante, algunos estudios obtienen peor resultado usando únicamente adjetivos que usando unigramas [4]. Esto indica que no solo los adjetivos proporcionan información útil, sino también otros tipos de palabras como los verbos ‘gustar’, ‘entristecer’, etc.

Posición: Otro experimento llevado a cabo por (B. Pang, L. Lee, 2002) fue el de anotar la posición de la palabra dentro del documento dependiendo de si se encontraba al inicio, mitad o final del texto. Sin embargo, los resultados no difirieron mucho de los obtenidos con unigramas.

Word2Vec: se trata de un conjunto de modelos creados por T. Mikolov en Google en 2013, utilizados para crear un mapeo de palabras a vectores n-dimensionales llamados *word embeddings*. Estos modelos consisten en redes neuronales poco profundas de dos o tres capas que se entrenan con un gran corpus de texto y producen un espacio vectorial de dimensiones reducidas. Así pues, las palabras que se suelen encontrar juntas en el corpus se representan a menor distancia que otras palabras menos frecuentes en el nuevo espacio vectorial n-dimensional.

Esta técnica utiliza los modelos CBOW (*Continuous Bag Of Words*) o *Skip-Gram* para crear una representación distribuida de las palabras. CBOW pretende predecir una palabra objetivo de acuerdo con una serie de palabras contextuales, mientras que con Skip-Gram se intenta predecir las palabras contextuales a partir de otra palabra. (Wikipedia, 2019)

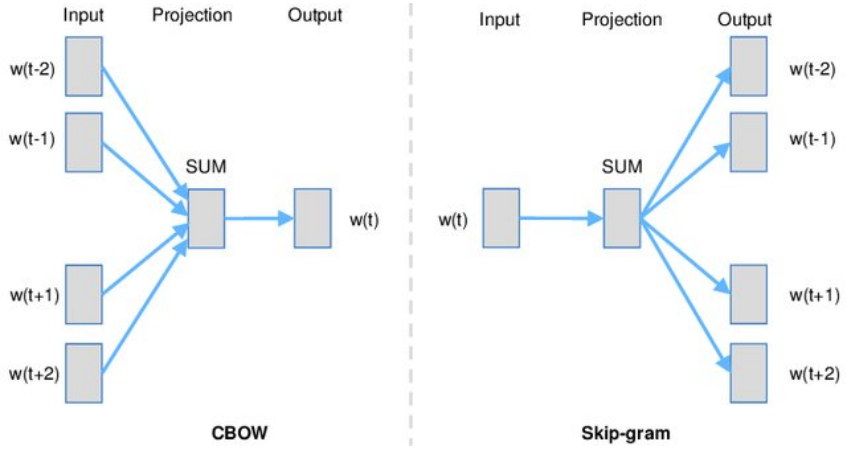


Figura 5: Modelos CBOW y Skip-gram. Fuente:https://www.researchgate.net/publication/326588219_Extending_Thesauri_Using_Word_Embeddings_and_the_Intersection_Method/figures?lo=1

La utilidad de crear estos mapeos de palabras se verá posteriormente en la construcción de redes neuronales artificiales convolucionales, donde se utilizarán para crear la matriz 2D de entrada.

2.4 Reducción de dimensionalidad del vector de características

Uno de los objetivos del análisis de sentimiento es obtener un conjunto de características apropiado para el entrenamiento del modelo de clasificación [3]. La tarea de selección y extracción de características es una tarea fundamental ya que permiten reducir la dimensionalidad del vocabulario a la mínima necesaria, lo cual tiende a reducir el sobre ajuste en el modelo de clasificación [11], el tiempo de cálculo y el espacio necesario para almacenar los datos [14].

Podemos clasificar las técnicas de Reducción de Dimensionalidad (RD) en dos categorías:

I. RD por selección (RDS): se selecciona un subconjunto de términos T' del conjunto T en función de su relevancia dentro del corpus. Existen tres categorías de algoritmos: filtros, *wrappers*, y *embedders*.

Filtros basados en álgebra lineal y teoría de la información:
Las técnicas de filtro más usadas en la literatura de análisis de sentimiento son los algoritmos PMI (Pointwise Mutual Information), que ya se mencionó anteriormente, Chi-cuadrado (χ^2) e Information Gain.

- **PMI**: El PMI de un par de resultados x e y perteneciente a variables discretas aleatorias X e Y, cuantifican la diferencia entre la probabilidad de su coincidencia dada su distribución conjunta y sus distribuciones individuales, suponiendo independencia matemática [17].

$$PMI(x,y)=\log\frac{P(x,y)}{P(x)\cdot P(y)}$$

Así pues, la medida PMI calcula la probabilidad de que una determinada palabra (x) esté relacionada con una determinada categoría (y)

- **Prueba de chi-cuadrado χ^2:** La prueba del chi-cuadrado generalmente mide la falta de independencia entre un término (t) y una clase (c), y comparada entonces con la distribución χ^2 con un grado de libertad [12]. La fórmula es la siguiente:

$$X_{(x,y)} = \frac{D(AJ-BI)^2}{(A+I)(B+J)(A+B)(I+J)}$$

Donde D es el número de documentos total, A es el número de documentos con la clase (c) que contienen (t); B es el número de documentos que contienen (t) y no pertenecen a la clase (c); I es el número de documentos de (c) donde no aparece (t) y finalmente J es el número de documentos de otra clase donde no aparece (t).

- **Information Gain:** método basado en la teoría de la información que usa la entropía como medida de impureza. Usado generalmente en la optimización de Árboles de Decisión, mide la información que proporciona una variable aleatoria X acerca de una clase c_i. (Wikipedia, 2019) Se calcula de la manera siguiente [20][21]:

  Primero, se calcula la entropía total del conjunto de datos, donde C es el conjunto de las categorías $\{c_1, c_2, \ldots, c_i\}$:

$$H(C) = -\sum_i P(c_i) \log P(c_i)$$

  Posteriormente, para cada una de las características, se calcula la entropía del conjunto de datos una vez observada la característica X:

$$H(C,X) = -\sum_i P(x_i) \sum_j P(c_j, x_i) \log P(c_j, x_i)$$

  Finalmente, se calcula la diferencia entre ambas:

$$IG(C,X) = H(C) - H(C,X)$$

La ventaja de los algoritmos de filtrado es que son sencillos, rápidos e independientes del método de clasificación utilizado posteriormente. Sin embargo, no tienen en cuenta las interacciones entre características [18].

***Wrappers*:** los *wrappers* son técnicas que obtienen el suconjunto óptimo de características mediante el entrenamiento y evaluación de varios modelos de *Machine Learning* utilizando varios subconjuntos de características diferentes. Éstos métodos consideran las dependencias entre características pero tienden a sobre ajustarse al conjunto de entrenamiento y son de gran complejidad

computacional y temporal, además de depender de algoritmos de *Machine Learning* concretos.

Embedded: los métodos incrustados son los que están implementados dentro del mismo algoritmo de aprendizaje computacional. Los métodos más conocidos son Árboles de Decisión y los modelos lineales regularizados *Ridge* y *Lasso*, los cuales aplican una penalización L_2 y L_1 respectivamente sobre el vector de pesos en la función de coste.

II. RD por extracción (RDE): se sintetizan nuevas características mediante transformación o combinación de las originales, que pueden ser de tipo diferente a las características originales, para maximizar la efectividad. Técnicas en esta categoría son PCA (Principal Components Analysis) y LSA (Latent Semantic Analysis)

La RD también se puede clasificar en **local**, si el algoritmo se aplica de forma separada para cada una de las categorías, o **global** si se aplica sin discriminar entre categorías.

2.5 Medidas de evaluación

Las medidas de evaluación de modelos utilizados para la clasificación de la polaridad son las mismas que las usadas para evaluar cualquier modelo de clasificación: precisión (precission), exactitud (accuracy), recuento (recall) y F-score.

	Predicción negativa	Predicción positiva
Casos negativos	TN	FP
Casos positivos	FN	TP

Tabla 4: Clasificación de predicciones. Fuente: elaboración propia

En la tabla 4 podemos ver la clasificación de las predicciones dependiendo de la categoría verdadera del caso y la predicción realizada. Las siglas TN, FP, FN y TP (True Negative, False Positive, False Negative y True Positive en inglés) corresponden al número de casos negativos clasificados correctamente, casos negativos predichos como positivos, casos positivos predichos como negativos y finalmente casos positivos correctamente predichos, respectivamente.

Precisión: nos indica la tasa de predicciones correctas del total de predicciones positivas (sean correctas o no), o lo que es lo mismo, nos indica si el modelo clasifica bien los casos negativos. Se calcula de la siguiente manera:

La precisión penaliza $Precission = \frac{TP}{TP+FP}$ los falsos positivos y nos indica qué tan bueno es nuestro modelo a la hora de

predecir correctamente los casos de la categoría positiva ya que un elevado número de falsos positivos incrementa el denominador y por lo tanto el resultado total disminuye.

Accuracy: número de aciertos totales positivos y negativos. Esta medida nos indica lo bien que clasifica nuestro modelo en general. Se calcula de la siguiente manera:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Esta medida es útil cuando el número de casos positivos y negativos está equilibrado ya que de otro modo el resultado estaría condicionado por la clase más frecuente.

Recall: nos indica el número de casos positivos correctamente clasificados. Sirve para maximizar la cantidad de predicciones positivas correctas pero no penaliza cuando el modelo realiza predicciones positivas incorrectas (FP). Se calcula de la siguiente manera:

$$Recall = \frac{TP}{TP + FN}$$

Como hemos visto, cada medida vista hasta ahora nos sirve para maximizar un aspecto de nuestro clasificador y no podemos guiarnos solamente por una de ellas: una precisión alta nos indica que el modelo no genera falsos positivos pero no nos dice nada sobre la bondad a la hora de predecir casos positivos. Por otro lado, el recuento nos dice si el modelo clasifica bien los casos positivos, pero podría ser que no clasificase nada bien los negativos. Por eso se suele utilizar otra medida llamada *F-score*, que tiene en cuenta las dos anteriores:

$$F - Score = 2\,x\,\frac{Precission\ x\ Recall}{Precission + Recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

La medida **F-score** es una media armónica de la precisión y el recuento. Mientras que la media normal trata todos los valores con la misma importancia o peso, la media armónica da más peso a los valores bajos.

2.6 Técnicas de clasificación

Existen diversos enfoques para el análisis de sentimiento **[5,7,8]**:

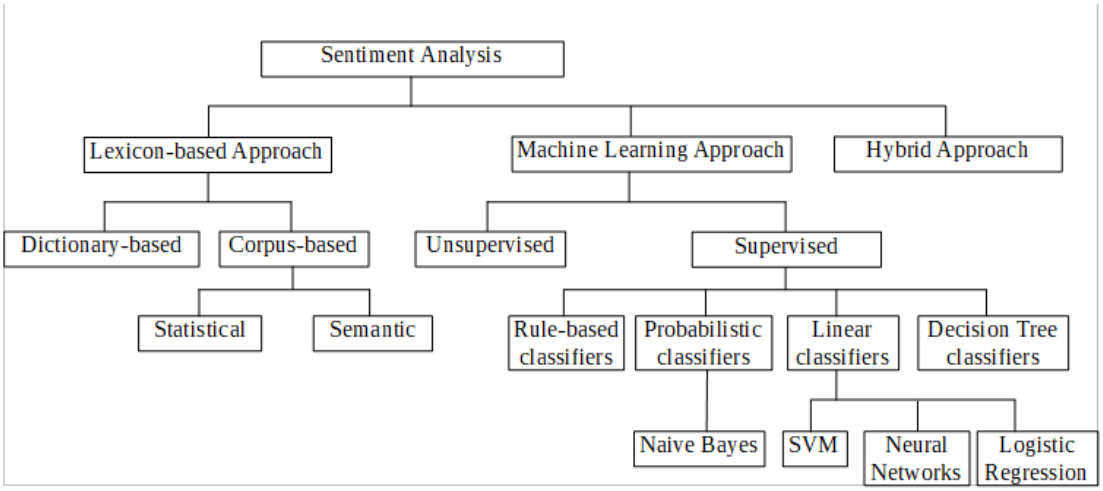


Figura 6: Clasificación de técnicas de clasificación

2.6.1 Enfoques basados en léxico

Las técnicas basadas en *léxico* (*lexicon* en inglés) emplean bases de datos de sentimiento o *sentiment lexicons* para obtener la polaridad de sentimiento de cada una de las palabras de opinión o expresiones de un texto [26].

Estos métodos son no supervisados ya que no es necesario disponer de textos previamente etiquetados con su polaridad.

Para obtener la orientación semántica de un texto, primero se lleva a cabo un preprocesamiento de éste tal y como vimos en secciones anteriores. Posteriormente se tokeniza el texto y para cada palabra se realiza una búsqueda de diccionario en alguna de las fuentes de léxico existentes. Los léxicos más utilizados son SentiWordNet, WordNet-Affect, MPQA y SenticNet [8].

En su aproximación más simple, las palabras con sentimiento se clasifican de forma binaria en positivas y negativas, aunque también existe una aproximación más sofisticada donde cada palabra tiene asignada un peso, convirtiendo los las categorías discretas en difusas (una palabra no es simplemente positiva o negativa, sino que puede ser muy positiva, poco positiva, poco negativa, etc.) [28]

El sentimiento total del texto se obtiene mediante agregación de las puntuaciones individuales de cada término, es decir, si una palabra tiene un sentimiento positivo se sumará el valor de dicho sentimiento al total; si es negativa, se restará. Finalmente, el valor agregado total se compara con un umbral determinado que separa ambas categorías; si supera el umbral el texto es positivo; en caso contrario es negativo.

Existen tres métodos para construir léxicos:

1. Construcción manual: este método es muy laborioso ya que hay que buscar todos los términos y asignarles una puntuación manualmente, por lo que prácticamente no se usa.

2. Basado en diccionario: se trata de un proceso iterativo en el que se parte de una semilla de adjetivos positivos y negativos, los cuales se buscan en WordNet para obtener sus sinónimos y antónimos. Los nuevos términos se añaden a la semilla, añadiendo sus sinónimos con la misma polaridad y los antónimos con polaridad invertida. El proceso se repite hasta que ya no quedan más palabras para añadir. Este método lo usa (M. Hu and B. Liu, 2004) en [27].

La desventaja de este método es que no es capaz de obtener la orientación sentimental de términos específicos de dominio como por ejemplo el uso del adjetivo 'pequeño', que puede ser positivo para un teléfono móvil pero negativo para un piso.

3. Basado en Corpus: el método basado en corpus utiliza un corpus de textos previamente etiquetados y se basa en las relaciones sintácticas halladas en éstos. Este método soluciona el problema que tienen los métodos basados en diccionario ya que es capaz de detectar la orientación sentimental de un término dentro de un dominio específico.

Se utilizan métodos probabilísticos y relaciones sintácticas para obtener la orientación semántica de las palabras a partir de una semilla. Dos técnicas que se utilizan son PMI (*Pointwise Mutual Information*) y LSA (*Latent Semantic Analysis*).

Otro método para detectar sinónimos y antónimos es mediante la localización de conjunciones y conectores como 'y', 'pero', 'sin embargo', y otros patrones sintácticos para decidir la orientación de los términos involucrados.

Uno de los autores más prominentes en el análisis de sentimiento no supervisado es (Turney, P., 2002) que desarrolló un algoritmo no supervisado denominado **PMI-IR** para clasificar *reviews* como 'recomendado' o 'no recomendado'. Este algoritmo emplea la combinación de PMI e IR para estimar la orientación semántica de una frase. Tras identificar en el texto pares de palabras que coincidentes con un determinado patrón de *Parts of Speech*, se calcula la orientación semántica de este par de palabras con la fórmula:

$$PMI(x,y)=\log\frac{P(x,y)}{P(x)\cdot P(y)}$$

Posteriormente se calcula la orientación semántica de la frase:

$$OS(frase)=PMI(frase,'excellent')-PMI(frase,'poor')$$

Se utilizan las palabras '*excellent*' y '*poor*' ya que en el sistema de puntuación de cinco estrellas son las palabras que comúnmente definen las *reviews* de cinco y una estrellas respectivamente.

Los valores de ambos PMI se estiman empleando un motor de búsqueda para hallar las coincidencias (*hits*) existentes de frase + ‘excellent’ y frase + ‘poor’. En el estudio se utiliza en buscador de Altavista porque proporciona el operador NEAR que restringe los resultados a aquellos en los que frase y palabra de referencia figuren dentro de una ventana de diez palabras, en cualquier orden. De las dos ecuaciones anteriores se puede derivar una tercera con algunas manipulaciones algebraicas :

$$OS(frase)=\log_2\left(\frac{hits(frase,'excellent')hits('poor')}{hits(frase,'poor')hits('excellent')}\right)$$

El último paso es calcular la media aritmética de la OS de todas las frases y clasificar la review como recomendada si esta es positiva y no recomendada en caso contrario.

2.6.2 Enfoques basados en *Machine Learning* (supervisados y no supervisados)

Los métodos de clasificación de *machine learning* se dividen en métodos supervisados y no supervisados, aunque la mayoría de métodos utilizados en en análisis de sentimiento caen dentro de la primera categoría [26].

Los métodos supervisados de *machine learning* utilizan dos conjuntos de datos etiquetados: un conjunto de entrenamiento, con el que se entrenan los modelos, y otro conjunto de prueba más pequeño con el que se evalúa el rendimiento del modelo ante casos desconocidos.

Las técnicas más utilizadas en la literatura del análisis son Naïve Bayes, Máxima Entropía, Máquinas de Vectores de Soporte y k-Nearest Neighbours, aunque desde hace relativamente poco se experimenta con algoritmos de aprendizaje profundo como las redes neuronales multicapa, redes convolucionales (CNN[3] en inglés) y redes neuronales recursivas (RNN[4] en inglés). Sobre estas últimas, algunos investigadores utilizan LSTM (Long Short-Term Memory) para solucionar el problema del desvanecimiento de gradiente.

De entre las técnicas de Machine Learning, (B. Pang, L. Lee, 2002) y (G. Sidorov et al, 2012) obtienen el mejor rendimiento utilizando SVM, seguido por Naive Bayes.

2.6.2.1 Naïve Bayes:

La técnica de Naïve Bayes es un método probabilístico supervisado que clasifica los textos basándose en la probabilidad de un vector de características de pertenecer a esa clase, asignándole la clase con mayor probabilidad.

Para hallar las probabilidades se utiliza un corpus etiquetado del cual se extraen las probabilidades de aparición de cada clase, es decir, el número de textos

3 CNN son las siglas de Convolutional Neural Network
4 RNN son las siglas de Recursive Neural Network

pertenecientes a cada clase, y para cada palabra del vocabulario el número de veces que esa palabra aparece en un textos etiquetados como positivo y cuántas en textos etiquetados como negativos. Con estas probabilidades se utiliza la Regla de Bayes:

$$P(c_i|d)=\frac{P(d|c_i)\cdot P(c_i)}{P(d)}$$

Figura 7: Regla de Bayes. Fuente: (Pang et al., 2002)

Donde c_i es la clase de la que queremos calcular la probabilidad dado el vector de características d, $P(c_i|d)$ la probabilidad a posteriori de c_i una vez observado d y $P(d|c_i)$ la probabilidad condicional de d una vez observado c_i. Desarrollando $P(d|c_i)$ la fórmula queda de la siguiente forma:

$$P(c_i|d)=\frac{(\prod_{i=1}^{m} P(d_i|c_i))\cdot P(c_i)}{P(d)}$$

Figura 8: Regla de Bayes extendida. Fuente: (Pang et al., 2002)

Donde x_i es cada una de las características del vector de características $d=\{d_1, d_2, \ldots, d_m\}$.

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, de ahí el calificativo '*naïve*' o ingenuo de su nombre, que es una suposición poco realista. No obstante, este método es efectivo debido a su robustez ante características irrelevantes y rinde bien en dominios con características con la misma relevancia [28].

2.6.2.2 Máxima Entropía:

Utilizado por (Pang et al., 2002), la técnica de Entropía Máxima es un clasificador probabilístico alternativo a Naïve Bayes que, bajo determinadas condiciones, supera el rendimiento de éste.

Este modelo se basa en el principio de Máxima Entropía. Su estimación de $P(c_i|d)$ toma la siguiente forma:

$$P_{ME}(c_i|d)=\frac{1}{Z(d)}\exp\left(\sum_{j=1}^{m}\lambda_{j,c}F_{j,c}(d,c)\right)$$

Donde Z(d) es una función de normalización, λ es el vector de pesos, donde un valor alto para λj,c indica que la característica xj del vector X es un indicador fuerte de la clase c, y Fj,c(X,c) es una función definida como:

$$F_{j,c}(d,c'):=\begin{cases} 1, & n_j(d)>0 \; y \; c=c' \\ 0 & en \; cualquier \; otro \; caso \end{cases}$$

El método de Máxima Entropía no realiza suposiciones sobre la independencia de las características, por lo que puede rendir mejor cuando no se da la condición de independencia entre las características, pero necesita más tiempo para entrenarse debido al problema de optimización de los parámetros lambda.

2.6.2.3 Support Vector Machines

El método Máquina de Vectores de Soporte (MVS) es uno de los métodos lineales más utilizados en problemas de clasificación de textos y que mejores resultados obtiene en comparación con los métodos anteriores, como demuestran los estudios de (B. Pang, L. Lee, 2002) y (G. Sidorov et al, 2012) .

El principal objetivo de esta técnica es hallar el hiperplano que mejor separe los elementos de dos clases dejando el máximo espacio o margen entre los puntos de datos más cercanos. Al maximizar el margen se consigue que haya menos incertidumbre sobre la clase a la que pertenecen los nuevos casos.

Los puntos de datos que se encuentran en los límites de dicho margen se llaman vectores de soporte.

La predicción de la clase se realiza calculando la función de decisión $w^T \cdot x+b = w_1x_1 + w_2x_2 + \ldots + w_mx_m + b$. Si la función tiene resultado positivo la clase predicha es la positiva, y si tiene resultado negativo se clasifica como la clase negativa. En caso de que la función sea 0 se trataría de un caso no concluyente. En resumen:

$$\hat{y}=\begin{cases} 0 & si \quad w^T \cdot x+b<0, \\ 1 & si \quad w^T \cdot x+b>0 \end{cases}$$

Figura 9: Función de decisión lineal de SVM.

El entrenamiento consiste en hallar el vector de pesos (w) que mejor clasifique las instancias del conjunto de prueba teniendo en cuenta a) que el margen tiene que ser máximo y b) si se permite que hayan instancias dentro del margen (margen 'blando' o *soft margin*) o no (margen 'duro' o *hard margi*n).

Un problema que tiene este método es que depende excesivamente de la selección del *kernel* apropiado para el dominio del problema ya que un *kernel* que funcione bien en un dominio no tiene por qué funcionar bien en otro [28] y que las observaciones tienen que ser linealmente separables.

Según (S. Vohra & J. Teraiya, 2013) SVM rinde mejor cuanto mayor es el espacio de características. [26]

2.6.2.4 K-Nearest Neighbours

La técnica k-Nearest Neighbours o K-vecinos más cercanos es una técnica de las llamadas *lazy* o perezosas puesto que no calcula ningún modelo matemático ni función lineal a partir de los datos sino que realiza las predicciones fijándose en los *k* puntos de datos a menor distancia. Por lo tanto no necesita ningún entrenamiento previo. Una desventaja de esta técnica es que es necesario disponer de todo el conjunto de datos para poder localizar los vecinos más cercanos y asignarle a la nueva observación la etiqueta que más se repite entre ellos.

El único hiperparámetro es el parámetro k. Valores grandes de éste evitan el efecto de ruido en la clasificación,

2.6.2.5 Árboles de decisión

La técnica de los árboles de decisión es una técnica supervisada de representación jerárquica usada principalmente para resolver problemas de clasificación [9][25].

Internamente, emplea métricas de impureza de Gini o Ganancia de Información para seleccionar aquellos atributos cuya presencia mejor discrimina entre clases.

En análisis de sentimiento, cada nodo intermedio del árbol se corresponde con la presencia o ausencia de un término, los cuales se van añadiendo al árbol desde la raíz por orden de importancia y donde los nodos hoja del árbol corresponden a categorías. Para predecir la polaridad de un nuevo texto simplemente hay que recorrer el árbol desde la raíz e ir descendiendo en función de si la palabra correspondiente a un nodo está presente o ausente en el texto.

También es utilizado como método *wrapper* de selección de características para su uso posterior por otros modelos de clasificación.

2.6.2.6 Redes neuronales

Las redes neuronales son modelos lineales supervisados cuya estructura intenta reproducir el funcionamiento de las neuronas biológicas y la forma en la que cada neurona se relaciona con el resto de neuronas formando entramados complejos.

A. Perceptrón simple

La arquitectura más básica de red neuronal es el perceptrón simple, que representa una única neurona. Esta arquitectura consta de una serie de n señales de entrada x_i conectadas a la neurona mediante n enlaces w_i, donde i=1,2,..,n, que tienen un peso sináptico asociado. La neurona calcula la función sumadora de cada valor de entrada multiplicado por el peso del enlace y posteriormente aplica una función de activación sobre el resultado de la suma, y.

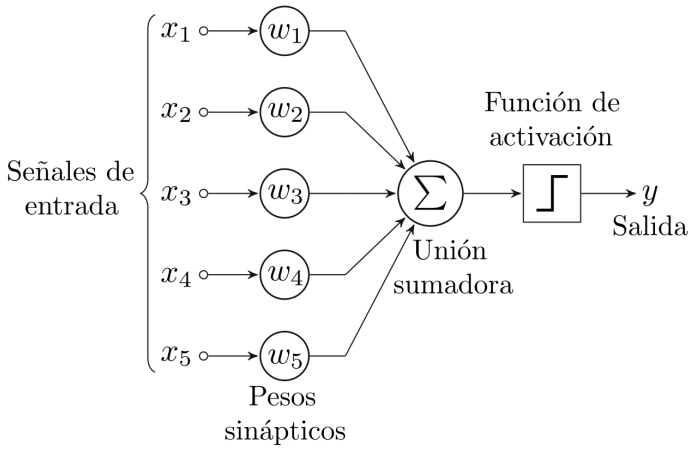


Figura 10: Estructura de un perceptrón simple. Fuente: Wikipedia

A partir de esta arquitectura simple, se han desarrollado a lo largo de los últimos años arquitecturas complejas que se alejan del modelo neuronal biológico con el objetivo de alcanzar soluciones más eficientes computacionalmente, tales como las redes neuronales convolucionales o las redes neuronales recurrentes, que veremos a continuación.

B. CNN

En esta arquitectura, cada neurona de la capa convolucional está conectada únicamente con un número de neuronas o señales de entrada que se ubiquen dentro de una ventana determinada. Cada neurona de la capa convolucional calcula la unión sumadora de ese subconjunto de neuronas de la capa anterior definido por un filtro de 2 dimensiones que se va desplazando por la matriz de la capa de entrada. Una ventaja de esta arquitectura es que se reduce el número de parámetros entrenables puesto que las dimensiones IxJ de los filtros son mucho más reducidas que el número de pesos NxM de dos capas contiguas.

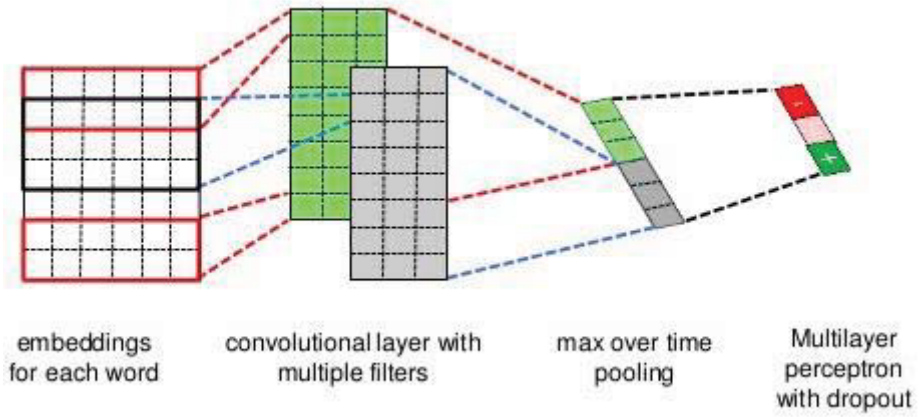


Figura 11: Representación de una red neuronal convolucional. Fuente: (Jain, K. Et al., 2018)

En el análisis de sentimiento, se ha utilizado esta arquitectura por varios autores como (Dos Santos, C. et al., 2014) [31] y (Pasupa, K. et al.,2019) [32] transformando el vector de características *bag-of-words* de dimensión m en una matriz de vectores n-dimensionales llamados *word embeddings* donde n << m y donde cada fila de la matriz corresponde a una palabra de la frase a analizar, alcanzando mejores resultados que los métodos SVM y NB.

(Dos Santos, C. et al., 2014) [31] propone el modelo CharSCNN con dos capas convolucionales que emplea representaciones a nivel de palabra, carácter y frase mediante *embeddings* de palabras entrenados con el modelo Word2Vec de (Mikolov et al., 2013) y que alcanza resultados de estado del arte analizando el sentimiento de *reviews* de películas y tuits con los corpus STS y SSTb, alcanzando una exactitud de 86,4% y 85,7% respectivamente utilizando dos clases (positivo/negativo).

Por su parte, (Pasupa, K. et al.,2019) [32] utiliza un modelo con solo una capa convolucional probando con 100, 200 y 300 filtros de dimensiones 3 x [dimensiones del vector de características], con un desplazamiento (*stride*) de 1. Los resultados contra un corpus de 1115 frases de texto en tailandés fueron de un 81,7% en F-score empleando *word embeddings*, etiquetado POS (modo *one hot encoding*) + Sentic.

C. LSTM RNN

Otro tipo de redes neuronales utilizadas en el análisis de sentimiento son las redes neuronales recurrentes y, concretamente, las LSTM (Long Short-Term Memory). Estas redes se caracterizan por utilizar neuronas que almacenan su estado interno durante un largo periodo de tiempo para procesar largas secuencias temporales de datos de longitud variable. Una ventaja de las redes LSTM es que solucionan el problema de desvanecimiento o explosión del gradiente que afecta a redes recurrentes convencionales, por lo que el

entrenamiento convergerá antes y detectará dependencias a largo plazo en las características [33].

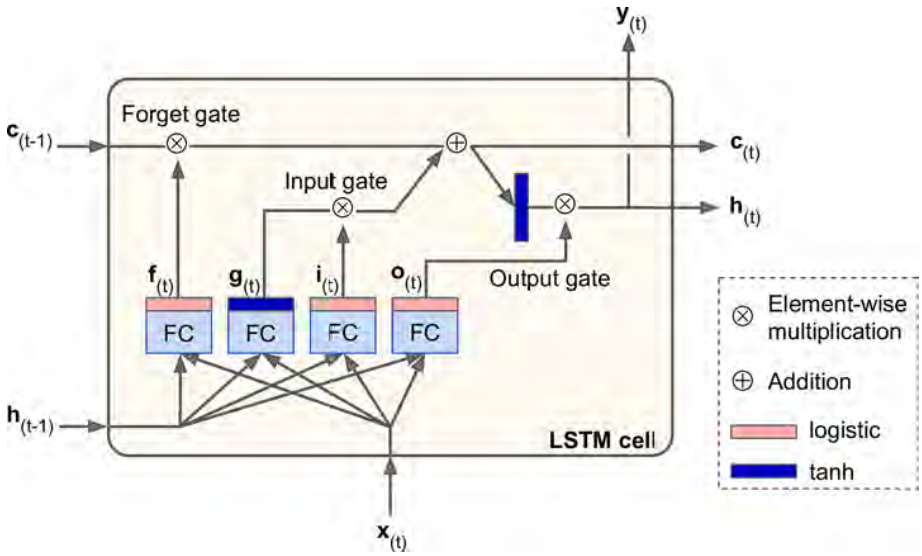


Figura 12: Célula LSTM. Fuente: [33]

En el esquema de la figura anterior, se aprecia que una célula LSTM está formada por cuatro capas completamente conectadas (FC o *Fully Connected*): f(t), g(t), l(t) y o(t). La capa g(t) se encarga de analizar el vector de entrada x(t) y el estado a corto plazo h(t-1) y envia el resultado de la función de activación tanh a la salida y(t) y. Esta es la única capa que encontramos en una célula básica. Las otras tres capas, con función de activación sigmoide, actúan como selectores de las puertas *Forget*, *Input* y *Output*. La capa f(t) decide qué partes del estado a largo plazo c(t-1) se borran; i(t) decide qué partes de g(t) se añaden nuevas; o(t) decide qué partes de c(t) se leen y se envían a y(t) y h(t).

Varios estudios comparativos como los de (Jain, K. Et al., 2018) y (Pasupa, K. et al.,2019) muestran que las redes neuronales CNN rinden mejor que las redes LSTM y LSTM bidireccional se erigen vencedoras sobre los métodos tradicionales de *machine learning*.

2.6.3 Técnicas híbridas

Algunos investigadores [34,35] sugieren la combinación de métodos de aprendizaje computacional con métodos basados en léxico para mejorar el rendimiento en la clasificación de sentimiento.

La principal ventaja de este enfoque es que se tiene los beneficios de ambos enfoques y se ha demostrado que mejora la exactitud. [6]

(A. Mudinas et al., 2015) desarrolla un modelo híbrido llamado pSenti que combina un léxico de sentimiento construido con 7048 palabras procedentes de recursos públicos, incluidos términos con comodines para una detección de de sentimiento inicial, con el método supervisado SVM, responsable de ajustar los valores iniciales de sentimiento, inicialmente en el rango [-3,3], así como ayudar a hallar nuevos términos [34].

2.7 Lecciones aprendidas del análisis del arte

2.7.1 Stemming vs Lematización

(Haryanto et al, 2018) muestra que el rendimiento del algoritmo SVM, usando un kernel RBF y validación cruzada 10-pliegues, es superior cuando se aplica *stemming* conjuntamente con reducción de dimensionalidad por Chi-cuadrado del 80%, alcanzando una precisión del 95% mientras que con lematización solamente se alcanza el 93%.

Los resultados de (Dave et al, 2003) [22] también concluyen que aunque con *Porter's stemmer* se obtiene una ligera mejoría en los resultados comparado con otras técnicas, los resultados son peores que usando unigramas sin normalizar.

La opinión general es que el *stemming* es mejor que la *lematización*, pero los resultados son muy variables y quizás dependen de otros factores: idioma, longitud del vocabulario, longitud de los textos, etc.

2.7.2 Uso de chi-cuadrado para reducir dimensionalidad

Del mismo estudio de (Haryanto et al, 2018) se desprende que las técnicas de reducción de dimensionalidad tienen un impacto notable en los resultados: aumentando gradualmente la tasa de reducción con Chi-cuadrado del 0% al 80%, pasamos de tener una precisión del 76% al 93% con lematización, y de un 84% a un 95% con *stemming*, confirmando los resultados conseguidos por (Emma Haddi et al.,2013) [16].

No obstante, el objetivo del análisis es la clasificación de textos según su temática, por lo que es posible que los resultados para un análisis de sentimiento no corroborasen esta conclusión.

2.7.3 Análisis de sentimiento vs análisis de temas

Por otra parte, (B. Pang, L. Lee, 2002) concluyen en su estudio realizado en 2002 [4] que las técnicas de *Machine Learning* como Naive Bayes, Maximum Entropy Classication y SVM no tienen tan buen rendimiento en el análisis de sentimiento como en la clasificación basada en temas, siendo Naive Bayes la peor, con una exactitud (*accuracy*) de 78,7% con unigramas y conteo de frecuencia, y SVM la mejor, con un 82,9%, también con unigramas pero con conteo de presencia.

La explicación es que, si bien es posible identificar el asunto de un documento simplemente analizando la ocurrencia de determinados términos, el sentimiento se puede y suele expresar de formas mucho más sutiles que, si bien los humanos son capaces de reconocer, suponen un reto para los algoritmos automáticos. No obstante, los resultados obtenidos por dichos algoritmos tienen un rendimiento superior al obtenido usando características seleccionadas manualmente por humanos mediante introspección.

2.7.4 Beneficios del preprocesamiento y los métodos de representación

Otro estudio realizado por (A.Krouska et al, 2016) [23] realiza una comparación de métodos de preprocesamiento y analiza su efecto en los resultados. Para el estudio utiliza TF-IDF como método de ponderación de las características, normalización de términos mediante el *stemmer Snowball*, eliminación de *stop-words y* tokenización con unigramas, bi-gramas y 1-3-gramas. Como métodos de selección de características examina la opción sin filtros, Information Gain > 0 y Random Forests con el 70% de los mejores resultados.
Finalmente utiliza los clasificadores NB, SVM, KNN y árbol de decisión C4.5.

Los resultados revelan que las técnicas de selección de características combinadas con un adecuado método de representación incrementan la exactitud de los modelos de clasificación. Especialmente, se destaca el uso de unigramas y 1-3-gramas, coincidiendo con la opinión de (B. Pang, L. Lee, 2002), que afirma que las mejores representaciones son los unigramas o bien una combinación de unigramas y bigramas. Otra conclusión a la que llega (A.Krouska et al, 2016) es que la exactitud es más alta con selección de características que sin filtros, salvo casos concretos.

(G. Sidorov et al, 2012) también realiza un extenso experimento sobre un corpus de tuits en español. En primer lugar, pone de manifiesto los problemas más comunes a la hora de tratar tuits en español como son el uso de jerga, abreviaciones (‘xfa’ en lugar de ‘por favor’), palabras mal escritas (pone el ejemplo de ‘maaal’ que se puede solucionar con eliminación de caracteres repetidos) o uso de extranjerismos (como ‘nice’). Estos autores proponen etiquetar nombres de usuario, emoticonos y URLs para simplificar los tuits, lo cual es inteligente ya que dichas palabras aumentan innecesariamente el vocabulario al no proporcionar información útil. También adopta POS-tagging y gestión de las negaciones mediante adición del prefijo ‘no_’ a la siguiente palabra, una solución diferente a la de (C.Musto et al,2014) que propone invertir la polaridad de toda la frase a partir de la negación.

El experimento utiliza los clasificadores NB, SVM y árboles de decisión C4.5 y J48. Los resultados indican que la precisión de SVM es la mayor entre todos los algoritmos, del 59%, usando tamaños de corpus entre 4500 y 7000 a intervalos de 500. También sale vencedor utilizando N-gramas con N entre 1 y 6, siendo los unigramas los que mejor precisión dan (61%), corroborando lo que ya habían demostrado (A.Krouska et al, 2016) y (B. Pang, L. Lee, 2002).

2.7.5 Importancia del dominio del corpus

(G. Sidorov et al, 2012) [24] muestra que el dominio del corpus con el que se entrena el modelo tiene un impacto enorme a la hora de clasificar textos que no pertenezcan a dicho dominio. En este caso emplea un corpus de opiniones sobre una marca de teléfonos con el que el algoritmo SVM consigue una precisión del 85,8% con textos del mismo dominio, pero solo un 28% con textos de otro dominio.

2.7.6 Frecuencia vs presencia

(Pang et al.,2002) demuestran que los vectores de unigramas basados en frecuencia muestran peor rendimiento tanto con Naive Bayes como con SVM, lo cual entra en contradicción con el estudio realizado por (McCallum & Nigan, 1998), donde tener en cuenta la frecuencia de las características sí tenía una relevancia significativa. No obstante, hay que tener en cuenta que el trabajo de (McCallum and Nigan, 1998) se centra en la clasificación basada en asuntos, por lo que es razonable pensar que la repetición de un término sí es relevante a la hora de clasificar un texto según el tema al que hace referencia, pero no tanto a la hora de extraer el sentimiento asociado al texto, menos cuando se analizan textos cortos como tuits.

2.7.7 Partes de la oración importantes

(Pang et al, 2002) demuestra que el uso de únicamente adjetivos como características no da buen resultado ya que se omiten otros tipos de palabras con un significado subjetivo como adverbios o verbos (‘encantar’, ‘gustar’, etc).

2.8 Retos del AS

Algunas de las particularidades del lenguaje natural a las que se enfrentan los investigadores de técnicas de análisis de sentimiento son la ironía, el sarcasmo, las diversas formas de negación, la ambigüedad del lenguaje y la multi-polaridad presente en los textos, consistiendo esta última en la presencia dentro del mismo documento de opiniones contradictorias como "me gustó mucho la película pero la actuación del protagonista fue horrible".

La solución a este último problema se puede resolver dividiendo la frase en dos y analizándolas por separado o bien analizando únicamente la primera frase y asignando una polaridad opuesta a la segunda frase que sigue la conjunción, aunque con ello se pierde exactitud a la hora de dar peso a la polaridad de la segunda frase ya que no se detecta en qué grado es ésta positiva o negativa.

Otro tipo de textos complicados de analizar son aquellos que requieren de información sobre el dominio ya que calificativos que se podrían considerar positivos en un dominio pueden ser negativos en otro. Por ejemplo, en ‘Este piso es pequeño’ el adjetivo ‘pequeño’ puede tener una polaridad diferente a ‘Este móvil es pequeño’. La solución a este problema es entrenar los modelos con datos pertenecientes al dominio en el que se pretende emplear.

3. Desarrollo del clasificador

En este capítulo detallaremos la metodología empleada para desarrollar el clasificador comenzando por la obtención del corpus, creación del vocabulario y preprocesamiento y normalización de los textos y terminando con la implementación de los modelos de red neuronal que serán dos: un perceptrón multicapa y una red neuronal recurrente utilizando una célula LSTM.

La metodología utilizada en este proyecto será la metodología CRISP-DM[5], una metodología iterativa utilizada ampliamente en proyectos de PLN (Procesado de Lenguaje Natural), que permite volver a fases del desarrollo anteriores en caso de ser necesario replantearse alguna de las decisiones tomadas.

Se usará durante todo el proyecto el lenguaje de programación Python por su sencillez, flexibilidad y su amplia comunidad de usuarios. Para la construcción de los modelos se utilizará la librería Keras.

Figura 13: Flujo de trabajo de un proyecto de PLN. *Fuente: https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72*

3.1 Elección del corpus

Se utiliza el corpus en español TASS (Taller de Análisis Semántico de la SEPLN[6]) General del año 2012 ya que contiene contiene miles de mensajes de Twitter en español debidamente etiquetados con su polaridad con una granularidad de 3 niveles: P (positivo), N (Negativo), NEU (Neutro) y NONE, categoría bajo la que se engloban todos aquellos mensajes que no contienen opinión.

Se elige este corpus ya que no existen muchos corpus en español con mensajes cortos de estilo parecido al que nos encontraremos en un chat de Twitch.

Otra opción que consideramos es la de descargar directamente tuits o mensajes del chat de Twitch en español utilizando la misma técnica que utilizan (Go, A., Bhayani, R., & Huang, L.,2009) [34] mediante el uso de la API de Twitter para obtener tuits y etiquetarlos como positivos o negativos según los emoticonos contenidos. No obstante, descartamos esta opción porque solamente tendríamos una clasificación binaria de los textos.

El primer paso que realizamos es la transformación de formato de los archivos del corpus, ya que están en formato .xml, por lo que antes de poder utilizarlo es

5 Siglas en inglés de Cross Industry Standard Process for Data Mining
6 Sociedad Española de Procesamiento del Lenguaje Natural

necesario convertirlo en un archivo separado por tabuladores para mayor comodidad y para poder cargarlos posteriormente utilizando la librería Pandas.

```
<tweets>
  <tweet>
       <tweetid>142378325086715906</tweetid>
       <user>jesusmarana</user>
       <content><![CDATA[Portada 'Público', viernes. Fabra al banquillo por
'orden'          del Supremo; Wikileaks 'retrata' a 160 empresas espías.
                 http://t.co/YtpRU0fd]]></content>
       <date>2011-12-02T00:03:32</date>
       <lang>es</lang>
       <sentiments>
       <polarity>
            <value>N</value>
       </polarity>
       </sentiments>
       <topics>
            <topic>política</topic>
       </topics>
  </tweet>
  ...
</tweets>
```

Texto 1: Extracto del archivo general-test-tagged-3l.xml. Fuente: http://www.sepln.org/workshops/tass/tass_data/dataset/general-test-tagged-3l.xml

La estructura jerárquica de los archivos es el siguiente:

Como se aprecia, el archivo contiene más información de la que necesitamos, por lo que solamente se extrae el valor de los elementos *content* y *polarity/value*.

Para la transformación utilizamos la librería *xml* de Python, recorriendo cada elemento <tweet> y almacenando en variables los valores que nos interesan.

```
import pandas as pd
import xml.etree.cElementTree as et

parsedTrain = et.parse( "corpus/general-train-tagged-3l.xml" )
parsedTest = et.parse( "corpus/general-test-tagged-3l.xml" )

def xml_to_pd(elementTree):
  dfcols = ['content', 'polarity']
  df_xml = pd.DataFrame(columns=dfcols)
  for node in elementTree.getroot():
      content = node.find('content')
  polarity = node.find('sentiments/polarity/value')
  df_xml= df_xml.append(pd.Series([content.text,
                  polarity.text],
                      index=dfcols),
                      ignore_index=True)
      return df_xml
```

Al finalizar el recorrido, se almacena el contenido de las estructuras en un archivo .tsv:

```
df_train = xml_to_pd(parsedTrain)
df_test = xml_to_pd(parsedTest)
df_train.to_csv('corpus/train.tsv', sep='\t', header=True)
```

```
df_test.to_csv('corpus/test.tsv', sep='\t', header=True)
```

A partir de este momento, solamente es necesario que cargar los datos desde dichos archivos usando la librería Pandas:

```
import pandas as pd
df_test = pd.read_csv('corpus/train.tsv', header=0, sep = '\t')
df_train = pd.read_csv('corpus/test.tsv', header=0, sep = '\t')
```

3.2 Análisis exploratorio del corpus

El corpus de entrenamiento de TASS contiene 60.798 textos divididos en 22.233 textos etiquetados como positivos, 15.844 como negativos, 21.416 sin sentimiento (NONE) y 1.305 neutros; el corpus de prueba contiene 7.219 textos repartidos en 2.884 textos positivos y 2.182 negativos, 1.483 sin sentimiento y 670 neutros.

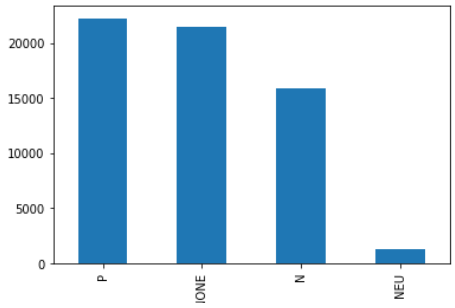


Figura 15: Distribución de textos por categoría del conjunto de entrenamiento

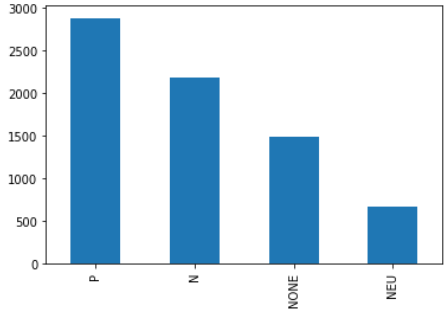


Figura 14: Distribución de textos por categoría del conjunto de prueba

Como se puede observar, la proporción de textos neutros es muy escasa en comparación con el resto, por lo que puede suponer un problema a la hora de obtener un modelo óptimo.

Se muestran a continuación los gráficos de barras correspondientes a la frecuencia de las 30 palabras más frecuentes en las categorías P y N:

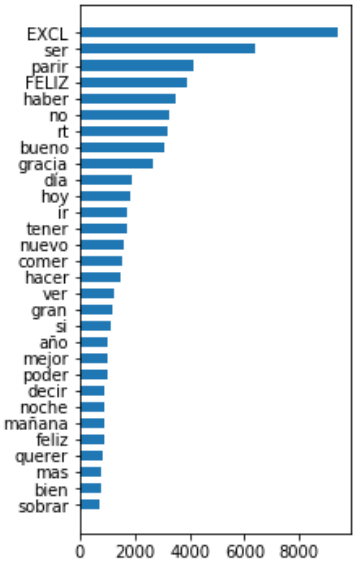


Figura 16: 30 palabras más frecuentes en cat. P

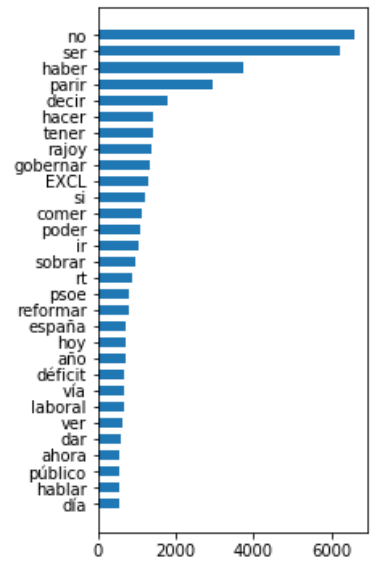


Figura 17: 30 palabras más frecuentes en cat. N

Y la de las categorías NONE y NEU:

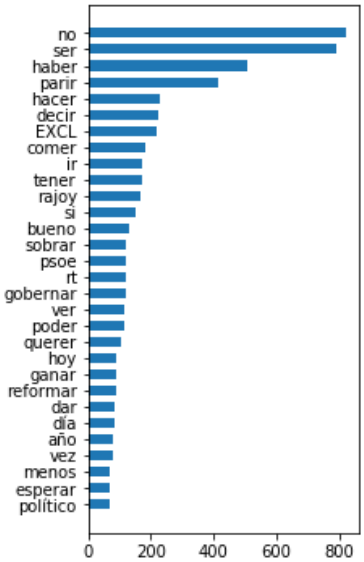


Figura 19: 30 palabras más frecuentes en cat. NEU

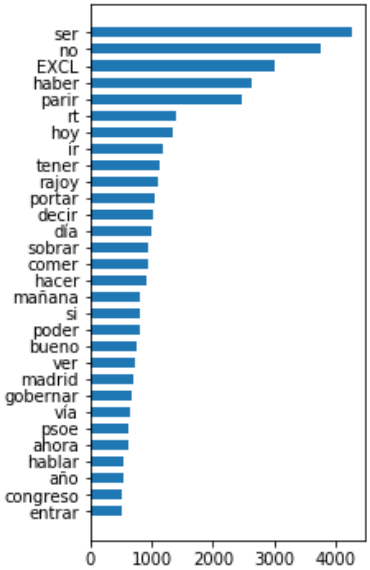


Figura 18: 30 palabras más frecuentes en cat. NONE

3.3 Preprocesamiento del corpus

Se llevan a cabo tareas de limpieza del corpus para transformar el corpus original en un conjunto de textos libres de elementos innecesarios para el análisis de sentimiento. Entre las acciones realizadas sobre el corpus se encuentran:

- **Conversión a minúsculas:** convertir el texto a minúsculas evita duplicidad de palabras al no distinguir entre letras mayúsculas o minúsculas.

- **Eliminación de URLs:** se eliminan las direcciones de la www que comienzan por 'http' ya que éstas no aportan información sobre opinión.

- **Eliminación de menciones:** se eliminan las palabras que empiezan por @ ya que generalmente se refieren a otros usuarios y no proporcionan información de utilidad.

- **Eliminación de *hashtags:*** se eliminan los *hashtags*, palabras que comienzan por # ya que en la plataforma donde vamos a aplicar el modelo no se utilizan.

- **Eliminación de RT:** se elimina la palabra RT de los retuits de Twitter.

- **Etiquetado de emoticonos:** se sustituyen algunos emoticonos como :-) ;-(:D, etc. por las etiquetas RISA, TRISTE o FELIZ.

- **Reemplazo de exclamaciones:** se sustituye el carácter '!' por la etiqueta EXCL ya que de otro modo se eliminarían en el paso siguiente.

- **Eliminación de caracteres especiales:** se elimina cualquier carácter que no sea alfabético (a-z y A-Z)

- **Eliminación de caracteres duplicados:** se eliminan secuencias de más de un mismo carácter consecutivo repetido. Esto ayuda a eliminar palabras enfatizadas como 'muuuy' o 'bieeen' transformándolas en sus versiones correctas 'muy' y 'bien'.

- **Eliminación de palabras de un solo carácter:** eliminamos abreviaciones de palabras de un solo carácter como 'q', que significa 'que' o 't', que significa 'te'. Este paso también ayuda a eliminar las palabras residuales que resultan del paso anterior, por ejemplo al reducir la palabra TT (*Trending Topic* en Twitter) a solamente 't'.

Una vez obtenemos un corpus 'limpio', procedemos a la **obtención del vocabulario** que utilizará nuestro modelo de clasificación.

3.4 Obtención del vocabulario

Con el corpus limpio, se crea el vocabulario mediante la cuenta de las ocurrencias de cada una de las palabras. Existen varias librerías en Python para contar elementos de vectores, como la clase *Counter* del módulo *Collections* o *CountVectorizer* de *Sklearn*. Sin embargo, nosotros realizaremos la cuenta sin utilizar estas librerías ya que necesitamos conocer las frecuencias separadas para cada una de las clases.

Se decide unir los conjuntos de entrenamiento y prueba a la hora de crear el vocabulario. De esa manera, el vocabulario será más amplio que utilizando únicamente el conjunto de entrenamiento.

```
corpus = pd.concat([df_test, df_train])
```

Para la creación del vocabulario es necesario llevar a cabo un proceso de tokenización o división de los textos en secuencias de palabras o tokens. Esto lo hacemos, para cada frase, con la función word_tokenze de NLTK de la siguiente manera:

```
tokens = nltk.word_tokenize(frase)
```

Seguidamente se eliminan las palabras de parada o *stop words*, que son aquellas palabras 'vacías' que no aportan ninguna información semántica. Se utiliza la lista de palabras *stop words* en español que proporciona el paquete NLTK[7] para eliminar las palabras incluidas en ella.

```
# Load stop words from the NLTK package
stop_words = set(stopwords.words('spanish'))
```

7 Natural Language Tool Kit.

Se pretende usar ***stemming*** y **lematización** para reducir las dimensiones del vocabulario ya que observamos que hay términos derivados que vienen a significar lo mismo como 'pierde', 'perdió' o 'perdáis', que se pueden agrupar bajo un solo término como 'perder' (usando **lematización**). Sin embargo, NLTK soporta *stemming* en español, pero no lematización. Una alternativa es instalar mediante el comando pip el paquete spacy, el cual sí soporta el idioma español.

Se decide comenzar creando un vocabulario lematizado. Se instala el modelo entrenado con ejemplos de noticias en español en su versión mediana con el comando: python -m spacy download es_core_news_md

Dentro del código, lo utilizamos de la siguiente manera:

```
import es_core_news_md
nlp = es_core_news_md.load()
```

A partir de aquí solamente tenemos que parsear una frase pasando como parámetro la frase:

```
tokens = nlp(preprocess(str(sentence)))
```

El objeto *tokens* devuelto contiene los atributos *token*, *lemma_* y *pos_* que contienen la palabra, el lema e información POS respectivamente, pero solamente utilizamos *lemma_* para acceder al lema de la palabra. Así pues, dejamos de utilizar el paquete NLTK para tokenizar las frases ya que spacy también lo hace. El código de la función es el siguiente:

```
def create_vocab(corpus, labels=['P','N','NEU','NONE'],
norm=None, stop_words =[]):

    vocab = {}
    spanish_stemmer = nltk.stem.SnowballStemmer('spanish')
    num_cats = len(labels)

    for sentence in corpus:
        if norm == 'lemma':
            tokenized_sentence =
  nlp(preprocess(str(sentence[0])))
        elif norm == 'stem' or None:
            tokenized_sentence =
  word_tokenize(preprocess(str(sentence[0])))

        for token in set(tokenized_sentence):
            if norm == 'lemma': word = token.lemma_
            elif norm == 'stem': word =
  spanish_stemmer.stem(token)
            else: word = token

            if word not in stop_words:
                if sentence[1] in labels:
                    cat_index = labels.index(sentence[1])
                    if word not in vocab.keys():
                        vocab[word] = [0]*num_cats
                        vocab[word][cat_index] = 1
```

```
                    else:
                        vocab[word][cat_index] += 1

    return vocab
```

Se recorre cada una de las filas del corpus añadiendo cada palabra filtrada y lematizada a un diccionario de Python y actualizando el número de veces que aparece en cada polaridad. La función anterior nos permite elegir el método de normalización que queremos utilizar entre 'lemma' (lematización), 'stem' (steming) o por defecto None, además de pasar como parámetro una lista de stop_words a eliminar, que por defecto está vacía.

Al final del proceso, conseguimos reducir un vocabulario inicial de más de 40 mil términos a 32.875 lematizando las palabras, donde la clave de cada entrada es la palabra misma y el valor un pequeño vector [f_P, f_N, f_{NEU}, f_{NONE}] que almacena el número de veces que una palabra aparece (por primera vez) en una frase por cada clase, ya que lo necesitamos para el siguiente paso.

3.4.1 Selección de características

Se realiza una selección de características mediante el método de **Ganancia de Información**, a partir de ahora GI, ya que el vocabulario todavía tiene un tamaño considerable.

Se opta por el método de GI y no por frecuencia absoluta ya que los términos más frecuentes no son siempre los que mejor discriminan entre clases, dado que un determinado término puede aparecer el mismo número de veces en documentos positivos y negativos, por lo que es evidente que ese término no aporta información. La fórmula para calcular la entropía H es la siguiente:

$$H = -\sum_{i=1}^{n} p_i \cdot \log_e(p_i)$$

Donde p_i es la probabilidad a posteriori $P(w_i|c_j)$ de la palabra *w* de aparecer en la clase c_i.

Finalmente se calcula para cada término la GI respecto a la Entropía inicial del conjunto de datos con la fórmula vista en el capítulo 2:

$$GI(C, X) = H(C) - H(C, X)$$

Donde C es el conjunto de clases C = {c_1, c_2, …, c_j} y X el subconjunto de textos donde aparece w_i.
Para lidiar con casos donde la probabilidad es nula, se opta por sumar a todas las probabilidades la cantidad de ajuste $1E^{-15}$.

Se seleccionan las palabras con IG > 0, es decir, aquellas que con su aparición aumentan la información y disminuyen la incertidumbre. Para el cálculo se utilizan las funciones propias siguientes:

```
def entropy(probs, adjust = 1e-15):
    total = 0
    for prob in probs:
        total += (prob + adjust)*np.math.log(prob+adjust,2)

    return -total

def IG(corpus_probs, word_weights, word_probs):

    corpus_entropy = entropy(corpus_probs)
    word_entropy = 0

        for i in range(len(weights)):
            word_entropy += (word_weights[i] * entropy(word_probs[i]))

    return group_entropy - word_entropy
```

Texto 2: Código de las funciones para calcular la Entropía e Information Gain

```
pos_count = corpus[corpus['polarity']=='P'].count()
  neg_count = corpus[corpus['polarity']=='N'].count()
  neu_count =  corpus[corpus['polarity']=='NEU'].count()
  none_count = corpus[corpus['polarity']=='NONE'].count()

  tot_cases = corpus['polarity'].count()
  class_counts = [pos_count[1], neg_count[1], neu_count[1],
    none_count[1]]
  class_probs = np.array(class_counts) / tot_cases

  for word in vocab.keys():

      wc1 = sum(vocab[word])# number of occurrences of word
       wc0 = tot_cases - wc1 # number of documents where word is
                 absent

      #probabilities of word taking value 1 for each class
              probs_1  =  [  vocab[word][i]  /  wc1  for  i  in
range(len(vocab[word])) ]
#probabilities of a word P(word=1|class)

      #probabilities of word taking value 0 for each class
       probs_0 = [ (class_counts[i] - vocab[word][i]) / (tot_cases
- wc1) for i        in range(len(vocab[word])) ] #probabilities of
a word P(word=0|class)

      # P(word=1)
      p_word = wc1 / tot_cases

    # P(word=0)
    p_abs_word = (tot_cases - wc1)/ tot_cases

    # IG(class_probs, values_weights, conditional_value_probs)
     vocab[word] = IG(class_probs, [p_word, p_abs_word], [probs_1,
  probs_0])
```

Tras obtener el diccionario de palabras con su correspondiente puntuación IG, se seleccionan las mejores 10000 palabras.

Por conveniencia y puesto que será necesario utilizar el mismo vocabulario empleado en el entrenamiento a la hora de realizar predicciones, guardamos el vocabulario reducido en un archivo de texto v*ocab_10000_4cat_lemma.txt.*

3.5 Primer modelo neuronal: red secuencial densa

El primer modelo que se crea es un clasificador binario empleando una topología de red neuronal secuencial densa[8] o perceptrón multicapa con una única capa oculta con función de activación ReLU y una capa de salida de 4 neuronas con función de activación Softmax.

Softmax es una función de activación que normaliza los resultados obtenidos por todas las neuronas de salida de forma que la suma de los resultados siempre esté en el rango [0.1]. Esto es especialmente útil cuando las categorías son disjuntas y excluyentes, como es el caso de las polaridades de sentimiento. La fórmula para calcular los valores de activación es la siguiente:

$$Softmax(x) = \frac{e^x}{\sum_{j=1}^{k} e^x}$$

También se usa Entropía Cruzada Categórica como función de coste, que es la función más popular para calcular el error en clasificadores multiclase, y método de optimización Adam, ya que ha demostrado ser una excelente alternativa al método de descenso de gradiente estocástico [35]. Se usa el valor por defecto para el factor de aprendizaje o learning rate del optimizador Adam de 0.001.

3.5.1 Obtención de los vectores de características

Debido a que la entrada de la red neuronal tiene que ser un vector numérico y de acuerdo con los resultados obtenidos por (B. Pang, L. Lee, 2002), utilizaremos inicialmente una representación ***bag of words*** y cuenta binaria o cuenta de presencia como peso de las características. Para la obtención del vector de características se usa una función creada para la ocasión que permite preprocesar y normalizar los textos durante el proceso.

```
def sen2vec(sentence, vocab, norm=None):
  # vectorizes a sentence into array of 1s and 0s according to
  vocabulary vocab and returns list.
  sentence_vector = []
```

8 También conocida en inglés como *fully connected* o completamente conectada ya que cada neurona de la capa se conecta con todas las neuronas de la capa anterior.

```
    sentence_tokens = []

    if norm == 'lemma':
        for token in nlp(preprocess(str(sentence))):
            sentence_tokens.append(token.lemma_)
    elif norm == 'stem':
        # load stemmer
        spanish_stemmer = nltk.stem.SnowballStemmer('spanish')

        for token in word_tokenize(preprocess(str(sentence))):
            sentence_tokens.append(spanish_stemmer.stem(token))
    elif norm == None:
        sentence_tokens = word_tokenize(preprocess(str(sentence)))

    #build vectors
    for word in vocab:
        if word in sentence_tokens:
            sentence_vector.append(1)
        else:
            sentence_vector.append(0)
    return sentence_vector

def vectorize(corpus, vocab, norm):
    # recorremos el corpus y vectorizamos todas las frases,
    añadiéndolas a un   vector.
    vectors = []
    for sentence in corpus:
        vectors.append(sen2vec(sentence, vocab, norm))
        return vectors

train_vectors=np.array(vectorize(df_train['content'], vocab,
norm='lemma'))
test_vectors=np.array(vectorize(df_test['content'], vocab,
norm='lemma'))
```

Se almacenan los vectores en sendos archivos train_vectors.npy y test_vectors.npy de tamaños 2.43 GB y 289 MB respectivamente, para su posterior recuperación.

3.5.2 Codificación de las etiquetas

Es imprescindible codificar las etiquetas categóricas ‘P’, ‘N’, ‘NONE’ y ‘NEU’ como números enteros mediante la clase *LabelEncoder* del módulo *Sklearn*:

```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
```

Si se utiliza la función de coste *‘binary_crossentropy’*, la codificación es la siguiente:

```
y_train = encoder.fit_transform(list(df_train['polarity']))
y_test = encoder.fit_transform(list(df_test['polarity']))
```

Otra opción es la función coste ‘categorical_crossentropy’, usada frecuentemente cuando se utilizan varias etiquetas de clase por muestra como por ejemplo ‘01’ o ‘10’. La codificación se realiza mediante la función to_categorical() de keras:

```
        encoder = LabelEncoder()
        y_train =
to_categorical(encoder.fit_transform(list(df_train['polarity'])))
        y_test =
to_categorical(encoder.fit_transform(list(df_test['polarity'])))
```

3.5.3 Creación del modelo

Una vez que tenemos las entradas preparadas como vectores *bag of words* y las etiquetas codificadas, se crea la arquitectura del modelo con 50 neuronas en la capa oculta:

```
model = Sequential()
model.add(Dense(50, input_dim=10000, activation='relu'))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               1000100
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 404
=================================================================
Total params: 1,000,504
Trainable params: 1,000,504
Non-trainable params: 0
_________________________________________________________________
```

Se usa ReLU por sus ventajas sobre la función sigmoide tales como la baja complejidad computacional en contraste con la función sigmoide, que necesita calcular exponenciales que son computacionalmente más costosos, y que evita el problema del desvanecimiento de gradiente que aparece en redes neuronales de varias capas cuando el valor agregado de la neurona es grande, al ser la derivada de la función sigmoide muy pequeña en sus extremos, lo cual hace que el proceso de convergencia[9] sea lento.

3.5.4 Entrenamiento y evaluación del modelo

Se entrena el modelo utilizando validación cruzada de 10 pliegues durante 10 épocas. Se muestra a continuación un gráfico de evolución de la exactitud y pérdida del entrenamiento y la validación en el que se alcanza una media global de 75,743 y un máximo de 76,536 con 50 neuronas.

9 El proceso de convergencia consiste en hallar un valor x tal que f(x) sea un mínimo local y por consiguiente f'(x)=0.

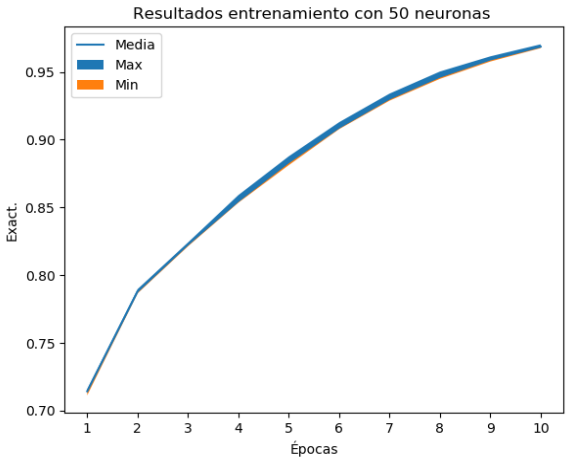


Figura 20: Evolución de exactitud durante en entrenamiento

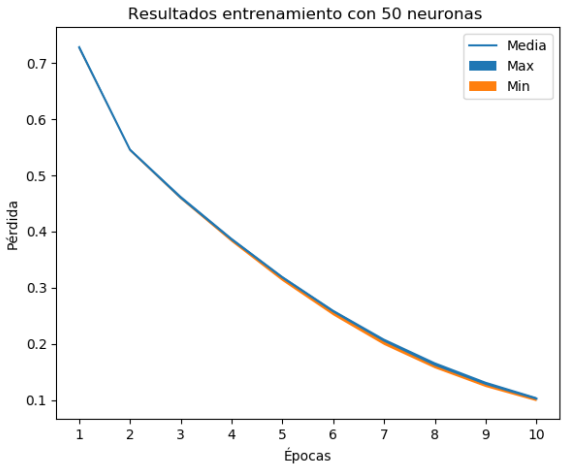


Figura 21: Evolución de pérdida durante entrenamiento

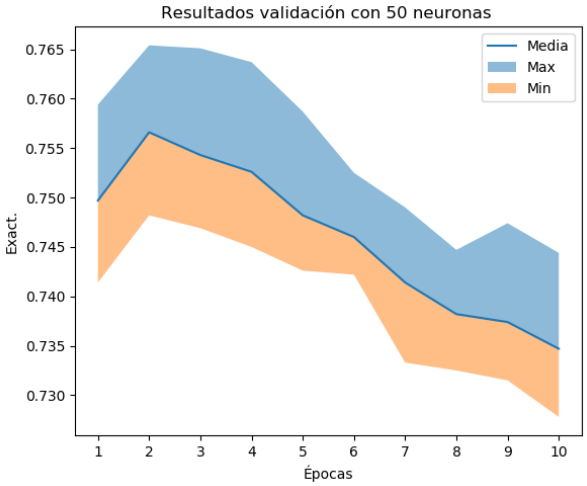


Figura 22: Evolución de exactitud durante la validación cruzada

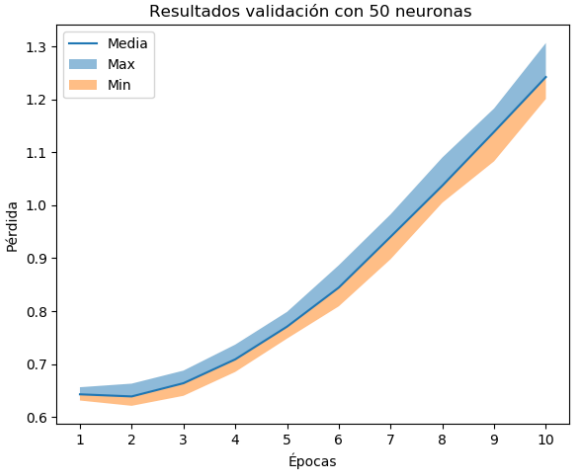


Figura 23: Evolución de la pérdida durante la validación cruzada

Debido a la dificultad para acertar exactamente con la época en que se va a obtener la mejor puntuación, se utiliza una función de *callback* proporcionada por Keras, *ModelCheckpoint*, que guarda en disco el mejor modelo cuando se realiza la validación si el resultado es mayor que el máximo histórico. La variable de *callback* se declara de la siguiente manera:

```
mc = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True,                                mode='max')
```

Y se indica en el atributo *callbacks* al llamar la función *fit* del modelo:

```
model.fit(padded_train, y_train, batch_size=20, nb_epoch=epochs,
verbose=1,     validation_data=(padded_test, y_test), callbacks=[mc])
```

Se utiliza el mejor modelo obtenido para evaluar el conjunto de prueba. El resultado obtenido es del 76,28% con la siguiente matriz de confusión:

	N	NEU	NONE	P	%
N	1683	4	343	152	77,13
NEU	210	173	130	157	25,82
NONE	43	0	1301	139	87,72
P	134	8	392	2350	84,48

Tabla 5: Matriz de confusión MLP con 50 neuronas

Tal y como habíamos anticipado en el análisis exploratorio de los datos, la tasa de acierto de casos de la categoría NEU es de menor del 26%, seguramente debido al hecho de que el corpus de entrenamiento solamente tiene 1305 casos neutros de 60.798.

Se realizan pruebas con 100, 150 y 200 neuronas en la capa oculta en las que se obtiene la mayor exactitud máxima con el modelo de 150 neuronas, por lo que se aplica un dropout de 0.5 tras la capa oculta para regularizar el modelo y comprobar si éste generaliza mejor. El valor de 0.5 se extrae del ejemplo *Multilayer Perceptron (MLP) for multi-class softmax classification* de la documentación de Keras.

Finalmente, se genera un vocabulario utilizando ***stemming*** en lugar de lematización. Los resultados con el vocabulario derivado son ligeramente inferiores en todas las puntuaciones, a las conseguidas con lematización, como se puede comprobar en la tabla 6.

A continuación se resumen los resultados obtenidos:

Tabla 6: Resumen de resultados MLP con 4 categorías

Neuronas capa oc.	**Exact. Media**	**Exact. Máx.**	**Exact. Conj. Prueba**	**Ac. N**	**Ac. NEU**	**Ac. NONE**	**Ac. P**
50	75,743	76,536	76,28	77,13	25,82	87,72	84,48
100	75,656	76,50	78,54	**80,47**	31,94	89,35	81,48
150	75,541	76,618	**79,13**	79,88	**33,43**	**89,95**	83,63
150 +DO	**75,976**	**76,727**	78,36	79,47	23,58	89,55	**84,5**
150 + DO (stemming)	75,946	76,698	77,46	77,91	21,04	89,28	84,15
200	75,700	76,397	N/D	N/D	N/D	N/D	N/D

```
model =Sequential()
model.add(Dense(150, input_dim=10000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Texto 3: Definición de modelo MLP con dropout de 0,5

No se aprecia una gran diferencia entre los resultados cuando se varía el número de neuronas, siendo posiblemente el mejor el modelo con 150 neuronas que sale ganador en cuatro de siete puntuaciones (sin aplicar *dropout*).

Se muestra la comparación durante las 10 primeras épocas de las puntuaciones medias y máximas de los 10 pliegues durante la validación cruzada, con y sin *dropout*:

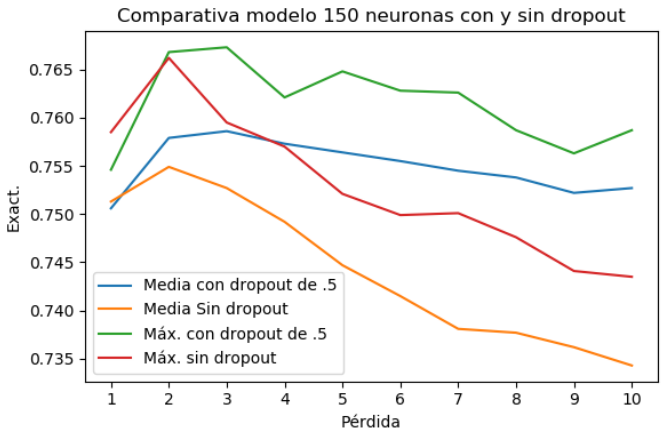


Figura 24: Comparativa de resultados con y sin dropout de modelo con 150 neuronas

Se observa una mayor estabilidad en los resultados máximos y medios (líneas verde y azul) cuando se utiliza *dropout* utilizando conjuntos de entrenamiento y validación diferentes, lo que sugiere que el modelo generaliza mejor y los resultados no empeoran tan rápido a medida que se entrena, por lo que queda claro el beneficio de utilizarlo a pesar de que no rinda especialmente bien con el conjunto de prueba estático usado.

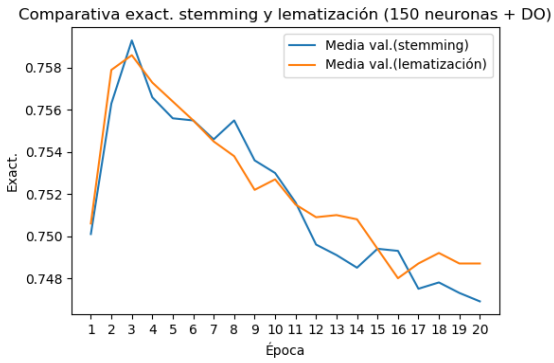


Figura 26: Comparación exactitud validación stemming vs lematización

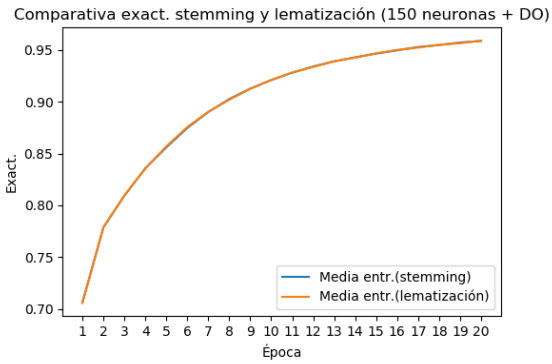


Figura 25: Comparación exactitud entrenamiento stemming vs lematización

3.5.5 Modelo MLP con 3 categorías: P, N y Otros

Vista la baja capacidad de los modelos anteriores para predecir casos neutros, se decide probar con un clasificador con tres categorías: P, N y Otros, incluyendo esta última categoría cualquier otro caso que no sea positivo ni negativo. Se entrena el modelo con la misma configuración que ha dado los mejores resultados: 150 neuronas en la capa oculta, usando lematización y *dropout* de 0.5 durante 5 épocas (suficiente en vista de la evolución de la exactitud en pruebas anteriores). Los resultados son, como se esperaba, mejores:

Tabla 7: Resumen de resultados MLP con 3 categorías

Neuronas capa oc.	**Exact. Media**	**Exact. Máx.**	**Exact. Conj. Prueba**	**Ac. N**	**Ac. Otras**	**Ac. P**
150+DO	**76,68**	77,521	86,00	84,23	85,88	87,45

	N	Otras	P	%Ac.
N	1838	247	97	84,23
Otras	110	1849	194	85,88
P	83	279	2522	87,45

Tabla 8: Matriz de confusión de la validación del conj. de prueba con modelo de 150 neuronas + dropout y 3 categorías con 10000 palabras

Dado que un modelo con tres categorías nos sirve perfectamente para el objetivo perseguido y en vista de la mejora de los resultados, decidimos adoptar esta solución en las siguientes pruebas.

3.5.6 Modelo con 3 categorías y dos capas densas

Para cerrar las pruebas con la arquitectura MLP, se añade una segunda capa de 100 neuronas para observar el comportamiento del modelo:

```
model = Sequential()
model.add(Dense(dim, input_dim=10000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(3, activation='softmax'))
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Tabla 9: Resultados de validación de modelo MLP con 3 categorías y dos capas densas

Neuronas capa oc.	**Exact. Media**	**Exact. Máx.**	**Exact. Conj. Prueba**	**Ac. N**	**Ac. Otras**	**Ac. P**

150+100 +DO	76,578	77,491	82,53	85,7	78,63	83,04

	N	Otras	P	%Ac.
N	1870	213	99	85,7
Otras	241	1693	219	78,63
P	160	329	2395	83,04

Tabla 10: Matriz de confusión de la validación del conj. de prueba con modelo de 150 + 100 neuronas + dropout y 3 categorías

3.6 Modelo de clasificador binario LSTM

El problema de la representación bag of words usada en el modelo anterior es que no se capturan las relaciones sintácticas entre palabras contiguas sino simplemente su presencia en la frase, por muy alejadas que se encuentren la una de la otra dentro del texto. Para solucionar este inconveniente y tratar de mejorar las predicciones se prueba una arquitectura secuencial recurrente empleando células de memoria LSTM (Long Short Term Memory), una mejora de las células RNN, las cuales realizan predicciones en base a las predicciones anteriores y que incorporan una memoria a largo plazo además de la memoria a corto plazo.

Se crea ahora un clasificador binario con la intención de ver si un clasificador binario de estas características es suficiente para detectar la polaridad neutra mediante la confianza de la predicción. Es decir, en caso de que la predicción devuelta tenga entre un 60 y un 40% de confianza, podríamos indicar que la predicción es neutra.

3.6.1 Creación del mapeo palabra : índice

Se crea un diccionario que mapea cada palabra del texto con el índice de ésta en el vocabulario puesto que será necesario para codificar los textos no como vectores de palabras sino como vectores de índices que apunten a sus correspondientes *embeddings:*

```
mapping = {word: i+1 for i, word in enumerate(vocab)}
```

Se comienza por el índice i+1 ya que necesitamos dejar libre la posición 0 para utilizarla para el relleno.

3.6.2 Codificación del corpus como vectores de índices

Se codifica cada texto del corpus como vectores de índices utilizando el diccionario anterior y se añade un relleno de ceros suficiente como para que

todas las secuencias de palabras tengan la misma longitud. Empezamos a referirnos a los textos como secuencias ya que es la terminología que se emplea en las redes neuronales recurrentes para referirse a las entradas sucesivas que componen una muestra.

3.6.3 Relleno / truncado

Se lleva a cabo el relleno de las secuencias de índices para obtener secuencias de longitud uniforme.

En la tabla 12 observamos que tras el preprocesamiento de los textos la longitud máxima se reduce de 146 a 22. Se considera un tamaño máximo de secuencia igual a la longitud máxima más un 50% para ir sobre seguro, quedando ésta en 33.

Se añaden ceros al principio de cada secuencia ya que preferimos dejar para el final la información útil y que la red LSTM no 'olvide' prematuramente. Las secuencias más largas de *max_seq_length* se truncan.

Tabla 11: Estadísticas de la longitud *de los textos antes de preprocesamiento*

count	*43143.000000*
mean	*111.863802*
std	*28.853678*
min	*6.000000*
25%	*95.000000*
50%	*122.000000*
75%	*136.000000*
max	*146.000000*

Tabla 12: Estadísticas de la longitud de los textos tras preprocesamiento

count	*43143.000000*
mean	7.478373
std	2.994328
min	0.000000
25%	5.000000
50%	7.000000
75%	10.000000
max	22.000000

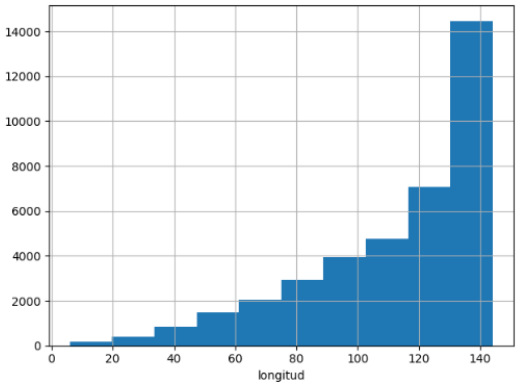


Figura 28: Histograma de longitud máxima (antes de preprocesamiento)

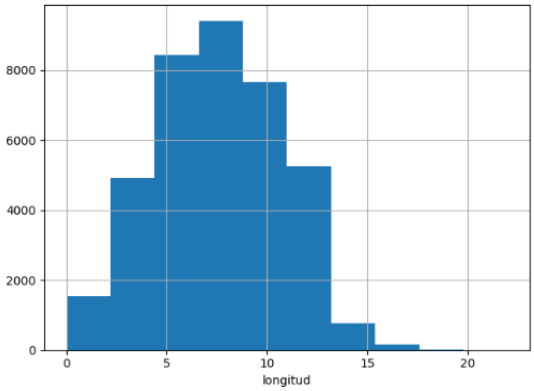


Figura 27: Histograma de longitud máxima (tras preprocesamiento)

Una particularidad de los modelos recurrentes es el formato de las entradas y salidas de cada capa. A diferencia de las redes neuronales densas y otros

modelos secuenciales donde la entrada tiene dos dimensiones (num_muestras, num_características), la entrada de la capa LSTM debe tener tres dimensiones

Por orden, las dimensiones de nuestros datos de entrada tienen que ser (muestras, pasos, características):

- **Muestras**: número de muestras de nuestro conjunto de datos. En nuestro caso, corresponde al número de frases del corpus. No es necesario indicarla ya que se calcula automáticamente.
- **Pasos**: número de observaciones que se realizan para cada muestra. En nuestro caso es un número indeterminado ya que cada texto tiene una longitud deferente.
- **Características**: número de características que tiene cada observación. En nuestro caso se trata del número de características de los embeddings, que suele estar en el rango 100-300.

En nuestro caso, la entrada debe tener una forma (43143, 33, 200) puesto que el corpus de entrenamiento tiene 43143 muestras (textos), cada muestra se compone de 33 pasos (palabras) y cada palabra tendrá 200 características que resultan de la transformación de cada palabra en *embeddings* de esas dimensiones.

3.6.4 Creación del modelo

Con el conjunto de datos listo y las etiquetas codificadas, se crea el primer modelo de clasificador binario empleando una célula LSTM:

```
    from keras.utils import to_categorical
    y_train =
to_categorical(encoder.fit_transform(list(df_train['polarity'])))
    y_test =
to_categorical(encoder.fit_transform(list(df_test['polarity'])))

    vocab_size = len(vocab)+1  # index 0 is for padding
    num_features = 200
    lstm_output = 100
    seq_length = 33
    epochs = 5

    model = Sequential()
    model.add(Embedding(vocab_size, output_dim=num_features,
        input_length = seq_length))
    model.add(LSTM(lstm_output, dropout_U=0.2, dropout_W=0.2))
    model.add(Dense(2, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])

    history = model.fit(padded_train, y_train, batch_size=20,
nb_epoch=epochs, verbose=5, validation_data=(padded_test,
y_test))
    score, acc = model.evaluate(padded_test, y_test, batch_size=20,
    verbose=2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 33, 200)           1211800
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               120400
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 202
=================================================================
Total params: 1,332,402
Trainable params: 1,332,402
Non-trainable params: 0
```

Figura 29: Estructura del modelo inicial LSTM_1

La topología del modelo consiste en una primera capa de *embeddings* que recibe como entrada un índice numérico que corresponde a uno de los términos del vocabulario y lo transforma en un vector de *n* características indicadas en el código con la variable *num_features*, que inicialmente establecemos en 200.

La capa Embeddings se conecta con la célula LSTM con *m* salidas indicadas por la variable *lstm_outputs*, configurada inicialmente en 100 uds.

Finalmente, la célula LSTM está conectada con una capa densa de salida donde obtendremos los resultados de las predicciones. En este caso, al ser el clasificador binario solamente necesitamos una neurona que se activará con valores entre 0 y 1.

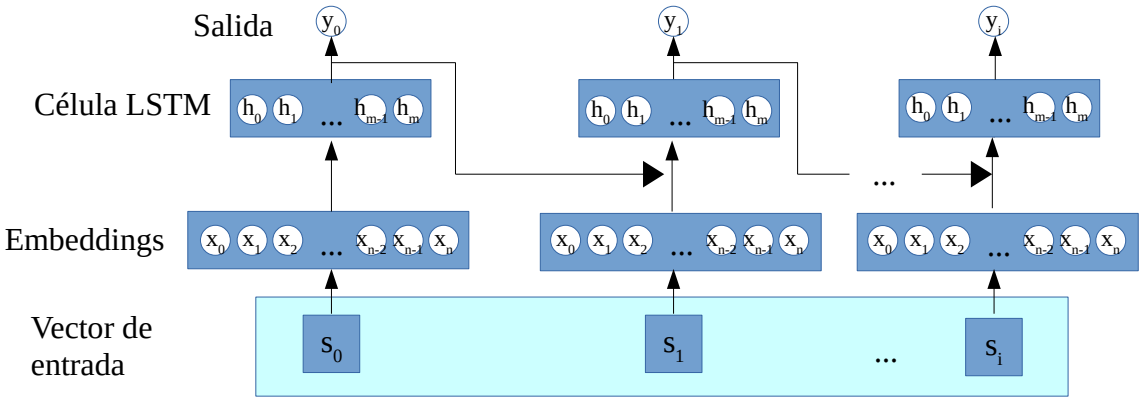


Figura 30: Esquema desplegado en el tiempo del modelo de una capa LSTM

El proceso se repite *k* veces por texto, donde *k* corresponde al tamaño de secuencia, que de momento hemos establecido en 33 pasos temporales (o palabras).

3.6.5 Entrenamiento y resultados

Se entrenan varias configuraciones del modelo variando los valores de varios hiperparámetros. En la tabla a continuación se muestran los resultados máximos de la métrica *accuracy* para varias configuraciones durante 10 épocas:

Tabla 13: Resumen de resultados máximos obtenido por el clasificador binario de una capa LSTM en validación de conjunto de prueba.

Caract. embeddings	Cross entropy	Dim. capa salida	Activación	Uds. salida LSTM				
				100	50	40	30	20
200	categórica	2	Softmax	87,03	86,79	86,75	87,13	86,89
100	binaria	1	Sigmoide	86,67	**86,73**	**86,95**	86,87	86,89
150	binaria	1	Sigmoide	**87,05**	86,89	86,73	**87,25**	**87,19**
200	binaria	1	Sigmoide	86,52	86,95	86,91	87,17	87,01

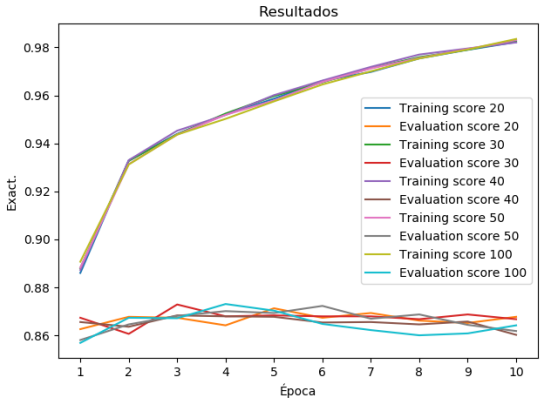


Figura 31: Resultados con varios valores de unidades LSTM de entrada y embeddings con 150 características

Como vemos, con una sola capa se obtiene el mejor resultado cuando la salida de la capa LSTM tiene un tamaño de 30 unidades de salida, usando activación sigmoide y un tamaño de salida de la capa de *embeddings* de 150, dentro de las primeras 3 o 4 iteraciones, en la que hemos conseguido una exactitud del 87,25%. En general, la exactitud no mejora a partir de dicha iteración sino que se deteriora, así como la pérdida.

Se lleva a cabo validación cruzada de 10 pliegues sobre el total de los 43.143 textos del corpus durante 10 épocas variando los valores de los hiperparámetros '*lstm_output*' (unidades de salida de la célula LSTM) y 'n*um_features*' (número de características de los *embeddings*). Se entrena el modelo con 38.828 muestras

y se valida con 4.315 de las muestras restantes. El resultado nos sorprende al ver que se consiguen exactitudes de más del 92%.

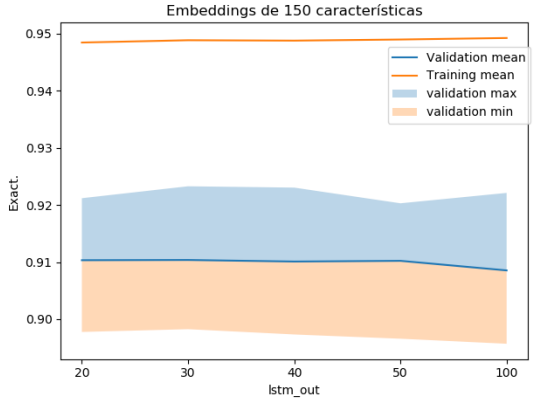


Figura 32: Resultados de exactitud máxima, media y mínima por valor lstm_out

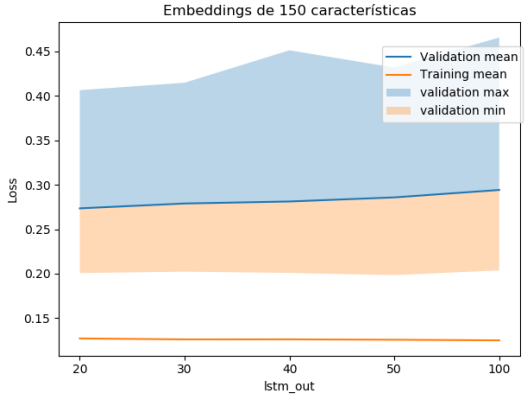


Figura 33: Resultados de pérdida máxima, media y mínima para cada valor lstm_output

Tabla 14: Resumen de validación cruzada

Métrica		lstm_output				
		20	**30**	**40**	**50**	**100**
Acc.	**Media**	0,94841	0,94881	0,94874	0,94894	0,94920
	Max.	0,97870	0,97924	0,97893	0,97821	0,98087
	Min.	0,87855	0,87942	0,88037	0,88074	0,88230
Val. Acc.	**Media**	0,91032	0,91036	0,91009	0,91021	0,90854
	Max.	0,92117	**0,92326**	0,92302	0,92028	0,92210
	Min.	0,89777	0,89826	0,89733	0,89659	0,89571
Loss	**Media**	0,12713	0,12606	0,12610	0,12567	0,12498
	Max.	0,28259	0,28007	0,27743	0,27527	0,27462
	Min.	0,05582	0,05404	0,05391	0,05493	0,04950
Val. loss	**Media**	0,27348	0,27896	0,28121	0,28582	0,29418
	Max.	0,40643	0,41484	0,45131	0,43187	0,46564
	Min.	0,20069	0,20228	0,20087	0,19837	0,20375

A continuación, se muestran las gráficas de los resultados de exactitud y pérdida en la validación de los 10 pliegues durante las 10 épocas de entrenamiento con num_features = 150, puesto que es el valor con el que se obtiene mejor resultado en la tabla 13 con lstm_output de 30. Se puede observar como se alcanza la mayor exactitud en la segunda época de 0.92326 con lstm_output = 30.

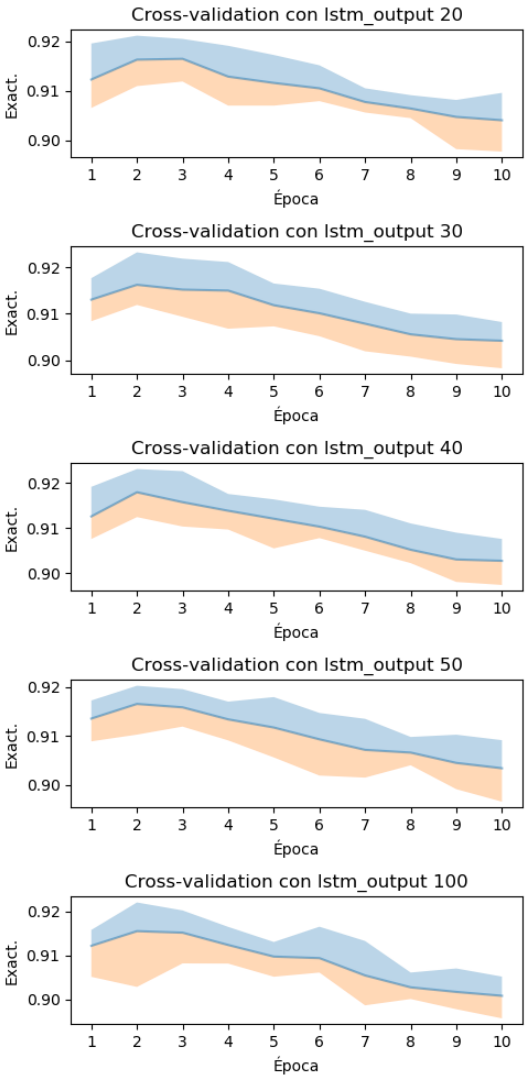


Figura 35: Exactitudes validación por valor de lstm_output

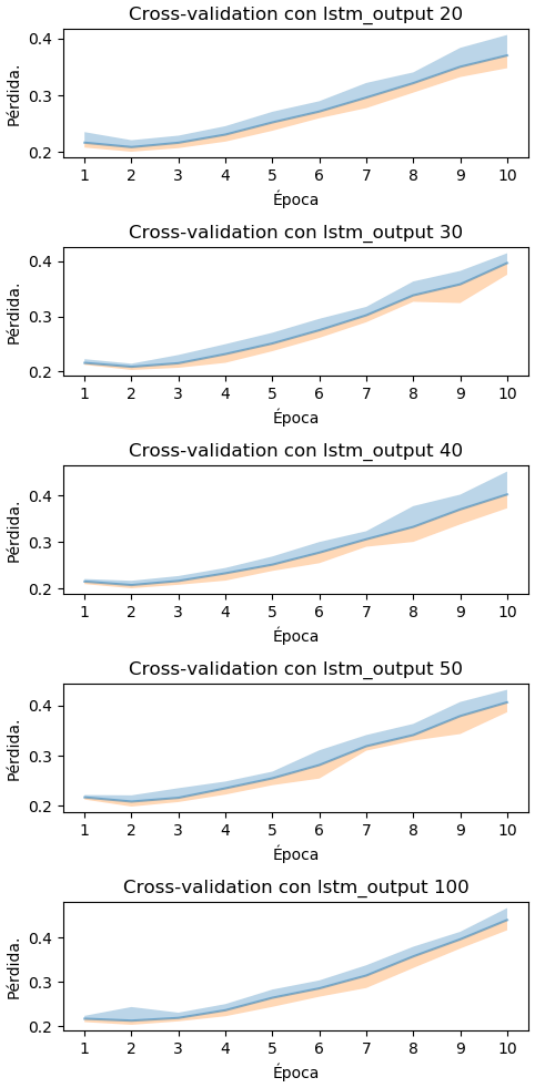


Figura 34: Pérdidas validación por valor de lstm_output

La pérdida de la evaluación también sufre un importante ascenso a partir de la época 3, por lo que a partir de esta información vemos innecesario realizar entrenamientos durante más de 3 épocas.

Así pues, de momento el modelo que mejor resultado da es el siguiente:

```
model = Sequential()
model.add(Embedding(6058, output_dim=150, input_length = 33))
model.add(LSTM(30, dropout_U=0.2, dropout_W=0.2))
model.add(Dense(1, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Respecto a los valores del entrenamiento, no se aprecia mucha varianza entre los valores de lstm_output. Tampoco se aprecia apenas diferencia entre los valores máximo, mínimo y medio para cada época. Aunque no lo parezca, en el gráfico se muestran esos tres valores, tan similares que solo se ve la línea azul. De nuevo, vemos como el mayor cambio en exactitud y pérdida se produce en las

dos primeras iteraciones, por lo que el modelo aprende bastante rápido y luego se ajusta demasiado a los datos de entrenamiento.

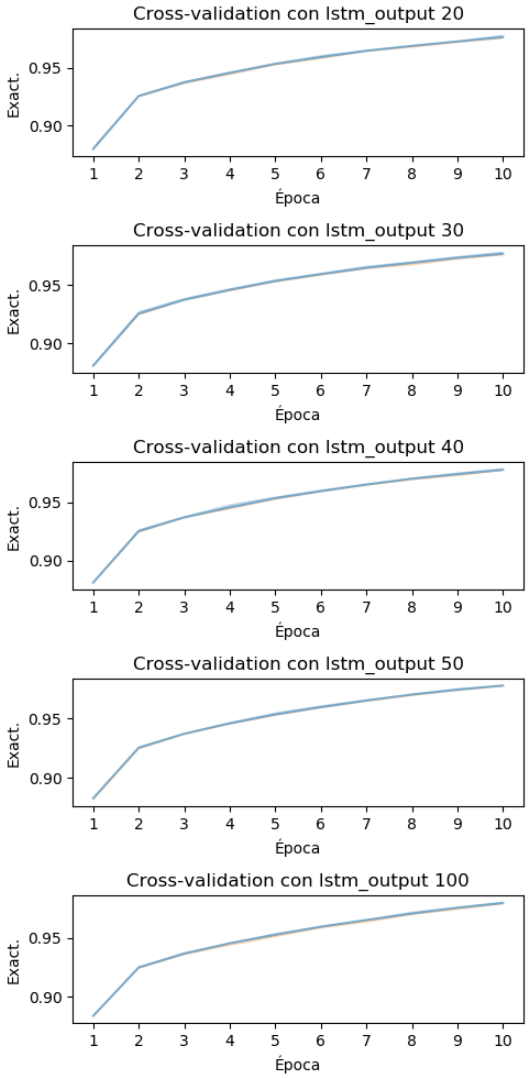


Figura 37: Exactitudes entrenamiento por valor de lstm_output

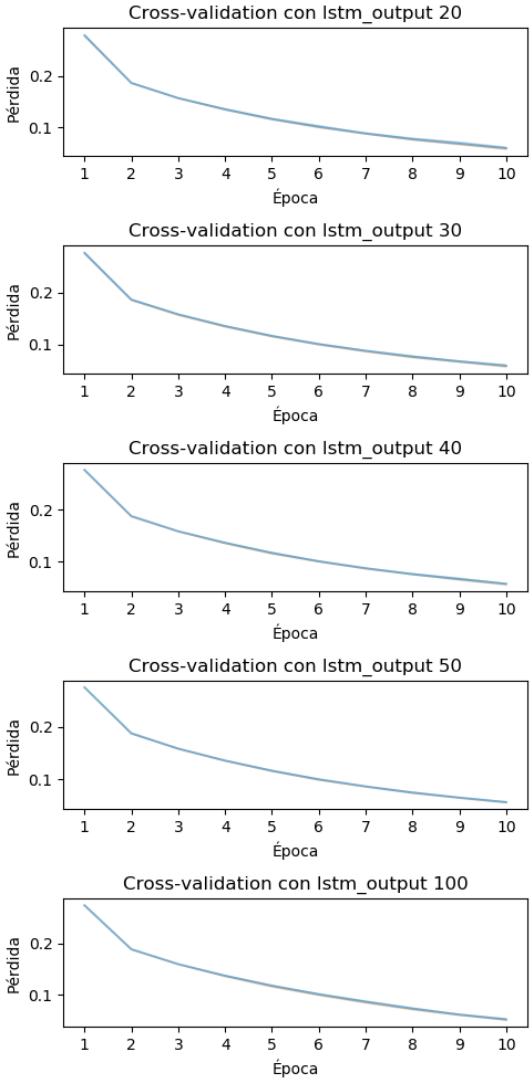


Figura 36: Pérdidas entrenamiento por valor de lstm_output

Se evalúa el mejor modelo obtenido, con una exactitud de validación media de 91,036% en la validación cruzada, con el conjunto de prueba y se obtiene un resultado de exactitud de 89,87%.

3.6.6 Efecto de los hiperparámetros dropout_U y dropout_W

Según la documentación[10] de Keras, las células LSTM tienen dos tipos distintos de dropout:

- **dropout_W**: número en coma flotante entre 0 y 1. Fracción de las unidades de entrada a descartar por las puertas de entrada.

10 http://faroit.com/keras-docs/1.2.1/layers/recurrent/#lstm

- **dropout_U**: número en coma flotante entre 0 y 1. Fracción de las unidades de entrada a descartar por conexiones recurrentes.

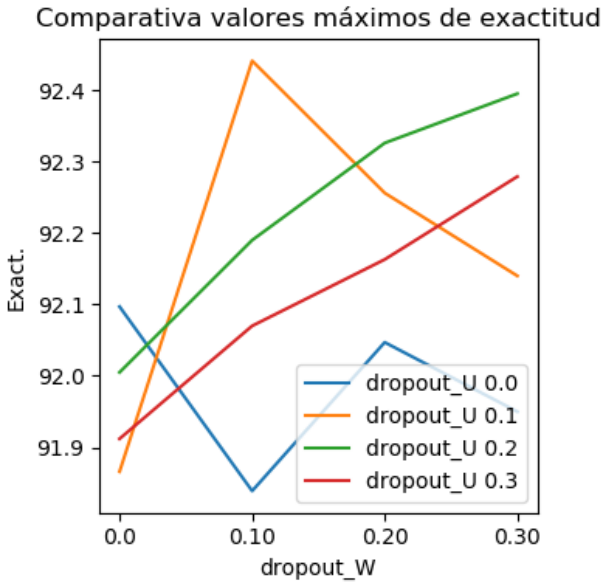
Los resultados anteriores se obtienen con un dropout_W y dropout_U de 0.20. Se intuye que si se disminuyen estos valores los resultados deberían empeorar, y al contrario si se aumenta ya que el modelo generaliza mejor. Se hacen pruebas durante 10 épocas de entrenamiento con los valores 0, 0.10 y 0.30 para ver si hay cambios significativos. El primer valor es el valor máximo obtenido en la validación cruzada; el segundo corresponde a la media obtenida y el tercer valor corresponde a la evaluación del conjunto de prueba. Se sombrean en verde los valores máximos obtenidos.

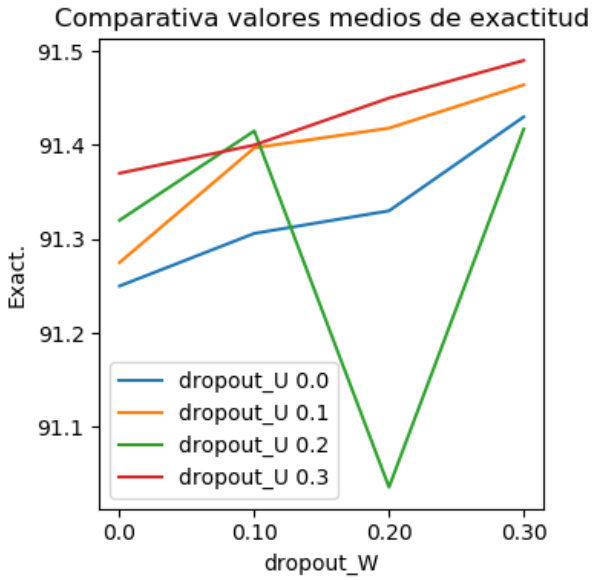
		dropout_W											
		0,0			0,10			0,20			0,30		
dropout_U	0,0	**92,097**	**91,25**	**91,610**	**91,839**	**91,306**	**90,32**	**92,047**	**91,33**	**92,53**	**91,95**	**91,43**	**92,30**
	0,10	**91,866**	**91,275**	**90,230**	**92,441**	**91,397**	**89,99**	**92,256**	**91,418**	**89,89**	**92,140**	**91,464**	**91,018**
	0,20	**92,005**	**91,320**	**90,288**	**92,19**	**91,415**	**90,327**	**92,326**	**91,036**	**89,87**	**92,395**	**91,471**	**91,117**
	0,30	**91,912**	**91,37**	**90,288**	**92,07**	**91,400**	**90,97**	**92,163**	**91,45**	**89,89**	**92,279**	**91,49**	**90,78**

Observamos una ligera correlación directa en la media de las exactitudes obtenidas en la validación cruzada cuanto mayor son los valores de *dropout*. Los valores máximos aumentan entre 0 y .20 pero luego vuelven a descender con 0,30.

Respecto al valor de dropout_W, también parece que en general mejora la media y máximo de las exactitudes cuanto mayor es este valor, aunque la diferencia tampoco es destacable.

Lo podemos ver más fácilmente en los dos gráficos siguientes:





Cada linea se corresponde con la evolución de un valor dropout_U según se varía el valor dropout_W (eje X).

Se llega a la conclusión de que los valores óptimos serían de 0.30 para ambos ya que proporcionan las medias más altas.

Tras probar el rendimiento de este modelo para predecir casos neutros a partir de la confianza de la predicción, se llega a la conclusión de que un modelo binario no nos es de utilidad ya que no se obtienen probabilidades en el rango esperado entre 0.6 y 0.4, siendo en ocasiones superiores a 0.75, por lo que se decide descartar esta opción.

3.6.8 Entrenamiento de la red LSTM con 3 niveles de sentimiento: P, N y Otras (NEU + NONE)

De la misma manera que se hizo en el apartado 3.5.4, se combinan los textos de las categorías NONE y NEU bajo la categoría O de 'Otros', con lo que obtenemos 22.721 textos en el corpus de entrenamiento y 21.53 en el de prueba bajo la categoría 'NEU'. Con ello se pretende entrenar el modelo para reducir el número de falsos positivos y negativos.

```
df_test[df_test['polarity']=='NONE'] = 'O'
df_test[df_test['polarity']=='NEU'] = 'O'
```

Se realizan pruebas utilizando el 100% del vocabulario, 32.431 palabras, así como otros tamaños de vocabulario entre 6000 y 10000 palabras a intervalos de 1000 usando lematización, tamaño de secuencia de 40, *embeddings* de 150 y lstm_output de 30. A continuación se muestra la tabla resumen de resultados de la validación cruzada y la validación del conjunto de prueba durante 10 épocas:

	Val. cruzada		**Val. Conjunto de prueba**		
Vocab	**Max**	**Media**	**Aciertos P**	**Aciertos N**	**Aciertos O**
32.431	-	75,978	0,78	0,74	0,71
10.000	**77,742**	**76,907**	0,83	0,78	0,78
9.000	77,201	76,466	0,83	0,81	0,75
8.000	77,565	76,616	0,82	0,79	0,78
7.000	77,404	76,533	0,82	0,81	0,73
6.000	77,315	76,786	0,81	0,74	0,71

Se observa que con 10.000 palabras se obtiene las mejores puntuaciones de exactitud máxima y media, por lo que consideramos este modelo como candidato para utilizarlo en la extensión.

3.6.9 Prueba con arquitectura de 2 capas LSTM con 3 niveles de sentimiento (P, N, NEU + NONE)

Se crea otro modelo con dos capas LSTM para comprobar si produce algún incremento en los resultados con las tres categorías utilizadas anteriormente:

```
lstm_output = 30
num_features = 150
seq_length = 33

model.add(Embedding(vocab_size, output_dim=num_features,
input_length = seq_length))
model.add(LSTM(lstm_output, dropout_U=0.3, dropout_W=0.3,
return_sequences=True))
model.add(LSTM(lstm_output, dropout_U=0.3, dropout_W=0.3))
model.add(Dense(3, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 33, 150)           1200150
_________________________________________________________________
lstm_1 (LSTM)                (None, 33, 30)            21720
_________________________________________________________________
lstm_2 (LSTM)                (None, 30)                7320
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 93
=================================================================
Total params: 1,229,283
Trainable params: 1,229,283
Non-trainable params: 0
```

Figura 38: Estructura del modelo de 2 capas LSTM

El parámetro *return_sequences = True*, tal y como se explica en la guía de Keras[11], indica que la primera capa LSTM debe devolver la secuencia completa de salidas generadas para pasárselas a la siguiente capa. En caso de no especificar ese parámetro se obtiene un error conforme la segunda capa está recibiendo un vector 2D en lugar de uno 3D.

Si nos fijamos en el resumen del modelo de una sola capa LSTM podemos ver que la forma de la salida (Output Shape) es (None, 30), ya que 30 es el tamaño de la salida de la capa LSTM, mientras que en el modelo de dos capas la forma de la salida es (None, 33, 30) puesto que se devuelven los resultados de los 33 pasos correspondientes a cada una de las palabras de la frase.

11 *https://keras.io/getting-started/sequential-model-guide*

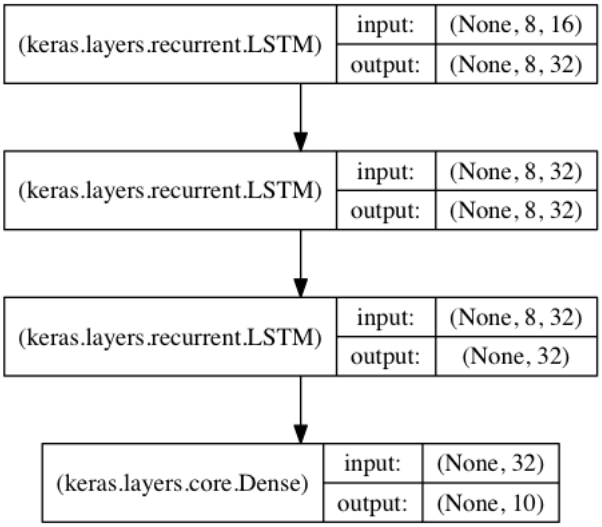


Figura 39: Ejemplo de capas LSTM apiladas. Fuente: https://keras.io/getting-started/sequential-model-guide

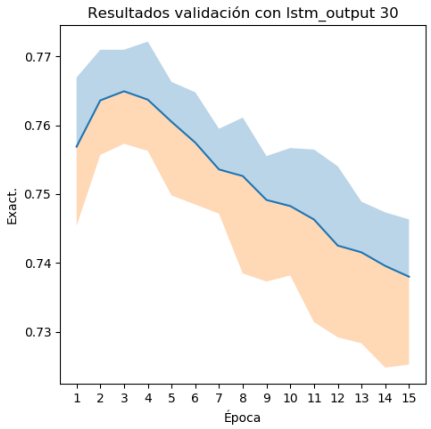
Se entrena el modelo durante 15 épocas utilizando un vocabulario de 10000 palabras con mayor IG generado con los textos de las categorías P, N y NEU+NONE, embeddings de 150, función de pérdida cross-entropy con 3 neuronas de salida, 30 uds. de salida para ambas células LSTM y activación *Softmax*.

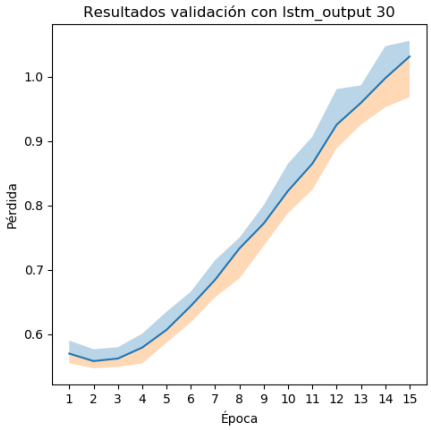
Tabla 15: Resumen de resultados de exactitud del modelo de dos capas LSTM con 3 categorías

Exact. media	**Exact. Máx.**	**Aciertos P**	**Aciertos N**	**Aciertos Otros**
76,903	77,4851	82%	77,61%	78%

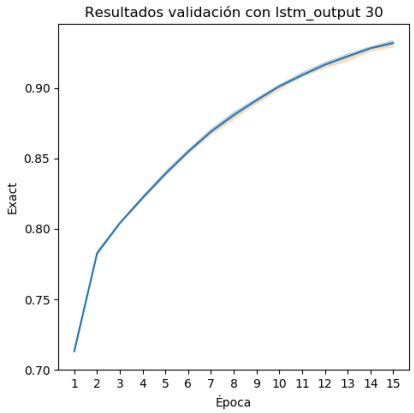
No se observa ninguna mejora al utilizar dos capas LSTM. Al contrario, los resultados han empeorado un poco respecto al mismo modelo de una sola capa.

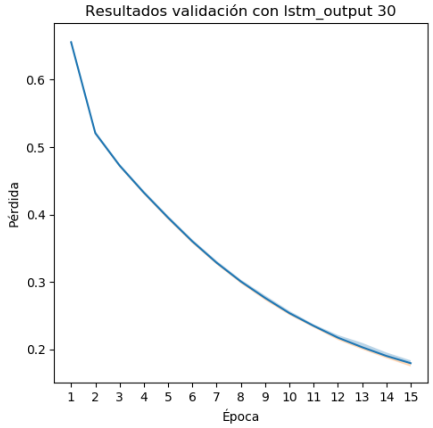
Se muestran a continuación los gráficos de exactitud y pérdida de la validación cruzada:





Y los de entrenamiento:





Con esta prueba damos por finalizada la fase de construcción y evaluación de los diversos modelos candidatos. A continuación se muestra un resumen de los resultados. No obstante, tal y como hemos visto, seleccionaremos el mejor modelo de clasificador de tres categorías: el modelo LSTM con vocabulario de 10000 palabras o bien el modelo MLP de 150 neuronas usando dropout y vocabulario también de 10000 palabras.

Tabla 16: Resumen de puntuaciones de modelos MLP, LSTM y CNN ordenados por media de exactitud con tres categorías.

Modelo	**Media**	**Máximo**
LSTM (vocab. 10000) 3 Cat.	76,907	77,742
LSTM 2 Capas 3 Cat.	76,903	77,485
LSTM (vocab.6000) 3 Cat.	76,787	77,315
MLP 150 + DO (Vocab.10000) 3 cat.	76,675	77,521
LSTM (vocab. 8000) 3 Cat.	76,616	77,536

MLP 150 + 100 + DO (Vocab.10000) 3 cat.	76,578	77,491
LSTM (vocab. 7000) 3 Cat.	76,533	77,404
LSTM (vocab. 9000) 3 Cat.	76,466	77,24
LSTM (100% vocab) 3 Cat.	75,978	n/d
MLP 150 +DO (Vocab.10000) 4 cat.	75,976	76,727
MLP 150 + DO (stemming) (Vocab.10000) 4 cat.	75,946	76,698
MLP 50 (Vocab.10000) 4 cat.	75,743	76,536
MLP 200 (Vocab.10000) 4 cat.	75,700	76,397
MLP 100 (Vocab.10000) 4 cat.	75,656	76,50
MLP 150 (Vocab.10000) 4 cat.	75,541	76,618

4. Análisis del estado del arte de las extensiones de Twitch para el análisis de sentimiento

Se lleva a cabo una búsqueda en Internet y el catálogo de extensiones de Twitch en busca de proyectos similares a los que estamos desarrollando en nuestro proyecto.

Vemos que existen ya extensiones que discriminan entre comportamiento positivo, neutro y negativo tales como:

1. Twitch-Chat-Snitcher (Anthony Miyoro)

URL:https://devpost.com/software/twitch-chat-snitcher

Esta extensión utiliza la librería *vaderSentiment* basada en investigaciones por Hutto, C.J. & Gilbert, E.E. (2014). titulada: VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text.

Ésta extensión recoge el sentimiento de cada uno de los comentarios del chat y calcula la media de todos ellos. No obstante, no discrimina por usuarios y tampoco penaliza comentarios muy negativos ya que al calcular la media las puntuaciones negativas se contrarrestan con las positivas, por lo que si un usuario escribe un comentario negativo y uno positivo el comportamiento será neutro. Tampoco se indica información nada sobre el reconocimiento de emojis, por lo que no se puede confirmar si los procesa.

2. Feels Chat (wlvs.tv)

Otro analizador de sentimiento que ofrece información más completa que el anterior es FeelsChat.

Además de informar sobre el sentimiento general del chat como positivo, negativo o neutro, esta extensión incorpora una gráfica lineal que muestra la evolución del sentimiento a lo largo de la retransmisión. También informa sobre el porcentaje de "*profanity*" o lenguaje incorrecto, lineas repetidas y spameo de emoticonos. No obstante, vemos que tampoco se mencionan usuarios concretos, por lo que es difícil saber qué usuario es el más negativo y cuál el más positivo.

3. Stream Hatchet Sentiment Gauge (https://blog.streamhatchet.com/tagged/sentiment-analysis)

Stream Hatchet Sentiment Gauge pretende analizar el sentimiento a nivel de usuario pero vemos que no hay ninguna versión lista.

Según el blog del proyecto, esta aplicación analiza el sentimiento de los comentarios del chat de Twitch y los presenta en una barra de puntuaciones del 0 al 10 mostrando a los autores enfrentados entre sí así como otras puntuaciones como la media de Twitch.

Un problema que vemos a este diseño es que, aunque el concepto es bueno, funciona con pocos usuarios pero cuando se desea mostrar una gran multitud de ellos la información puede volverse confusa,

4. Stream Sentiment Analysis (nyceane)

Durante el desarrollo del trabajo se añade al repositorio de extensiones esta extensión que muestra un gráfico de pastel para mostrar el sentimiento global del canal. Esta extensión tampoco discrimina el sentimiento entre los usuarios del chat, por lo que todavía podemos innovar en ese aspecto con nuestra extensión.

Como hemos visto, existe una oferta limitada de extensiones de Twitch para el análisis de sentimiento y de ellas ninguna nos permite actualmente ver el sentimiento de los espectadores ni realizar acciones de respuesta a estos comentarios. Una posibilidad sería banear usuarios muy negativos por un periodo de tiempo (5 min. por ejemplo) previo avisos automáticos. Para ello, la extensión tendría que incorporar un bot que avisara a los usuarios más conflictivos con un mensaje automático como "$usuario$, te recuerdo que no está permitido el lenguaje malsonante".

Otro de los aspectos en los que se puede innovar es crear una aplicación que funcione bien con el idioma español y/o catalán. Éste último sería una característica a implementar en el futuro.

5. Desarrollo de la extensión de Twitch

5.1 Conceptos previos

Las extensiones de Twitch son aplicaciones interactivas que se superponen en la ventana de video de la retransmisión o bien muestran bajo ésta.

Twitch implementa soporte para varios tipos de extensiones:

- **Panel**: se muestran bajo el área del reproductor de video. Permanecen activas aunque el canal esté inactivo.

- **Superposición de video**: Se muestran sobre la reproducción de video en forma de superposición transparente y solamente se muestran cuando la retransmisión está activa. Ocupa el 100% del área del video.

- **Componente de video**: se muestran por encima del video pero solamente ocupa una parte de éste. Estas extensiones también son visibles únicamente cuando la retransmisión está activa.

En general, la arquitectura de una extensión Twitch consta de las siguientes características:

- Aplicación cliente/servidor HTML + JS + CSS
- Tiene entre 2 y 3 *front ends* HTML
  - Visor (obligatorio)
  - Configuración (obligatorio)
  - Dashboard en directo (opcional)
- *Extension Backend service (EBS)*
  - Cualquier lenguaje de programación
  - Cualquier hosting
  - Usa cualquier almacenamiento de datos

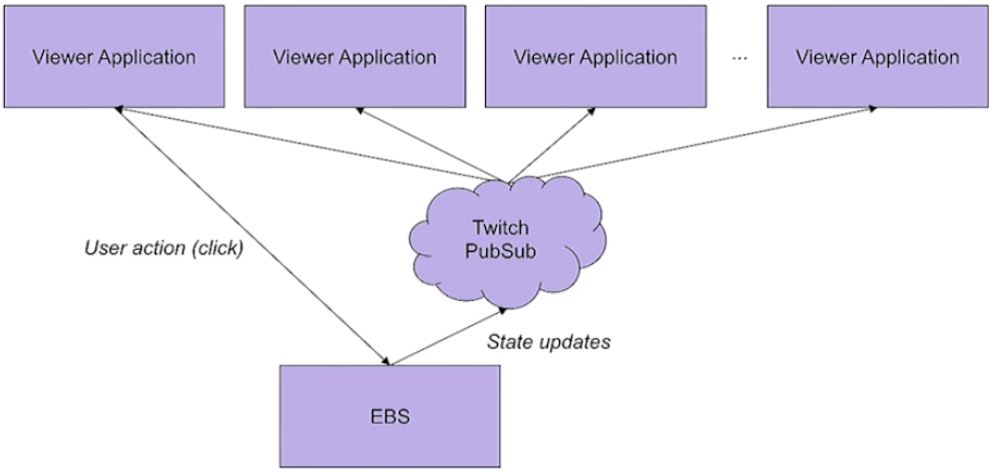
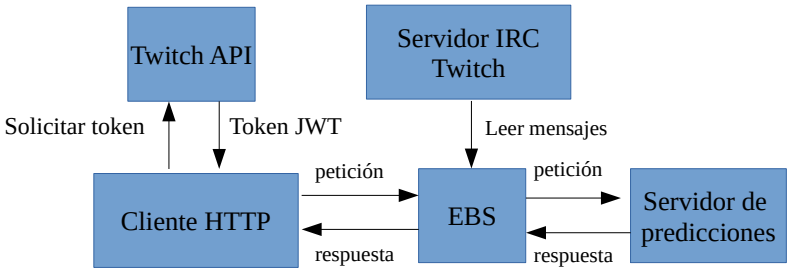


Figura 40: Arquitectura de una extensión Twitch. Fuente: Twitch.



Dibujo 1: Flujo de información de nuestra aplicación

El flujo de datos de la aplicación es la siguiente:

1. El cliente HTTP solicita un token JWT a la API de Twitch, el cual autoriza al cliente a comunicarse con el EBS mediante una petición HTTP GET y enviándole el token.

2. El EBS, tras verificar el token, abre una conexión con el servidor IRC de Twitch para leer los mensajes mediante sockets.

3. Tras la lectura y parseo de los mensajes, el EBS envía éstos al servidor de predicciones, donde estará desplegado el modelo de clasificación de polaridad de sentimiento, mediante una solicitud HTTP GET o POST.

4. El servidor de predicciones, tras analizar y clasificar los mensajes, devuelve las probabilidades de las predicciones en el cuerpo de la respuesta en formato JSON.

5. El EBS, tras recibir la respuesta, puede dar formato a los datos mediante HTML o enviarlos en formato JSON al cliente.

La comunicación entre los visores y el EBS se realiza mediante AJAX (Asyncronous Javascript And XML), y se envían *tokens* de autorización facilitados por Twitch.

La comunicación de las actualizaciones de estado del EBS no se realiza directamente entre éste y los agentes de usuario sino que se utiliza el servicio *Publish-Subscribe* PubSub de Twitch, que se encarga de actualizar la información para los cientos o miles de espectadores utilizando la aplicación. No obstante, esto solamente es cierto cuando hay que comunicar los cambios de estado a todos los clientes. En el caso de extensiones que solamente sean visibles para el retransmisor la conexión se puede realizar directamente.

5.2 Creación del frontend

Se utiliza la aplicación ***Extensions Developer Console***, proporcionada por Twitch. para realizar el desarrollo de la extensión, accesible desde https://dev.twitch.tv/console/extensions.

Se instala el ***developer rig*** **o plataforma de desarrollo de Twitch**, que nos permitirá desarrollar las extensiones y probarlas de forma rápida y local y se crea el proyecto Hello world para que la aplicación genere todos los archivos necesarios.

5.2.1 Creación de la primera extensión: Hello world.

Se crea el proyecto a partir del ejemplo Hello World que nos servirá de base para desarrollar el frontend.

En Developer Rig, se hace *click* en *Add Project → Create Project* en la esquina superior izquierda. Nos aparece un asistente de 3 pasos que tenemos que rellenar:

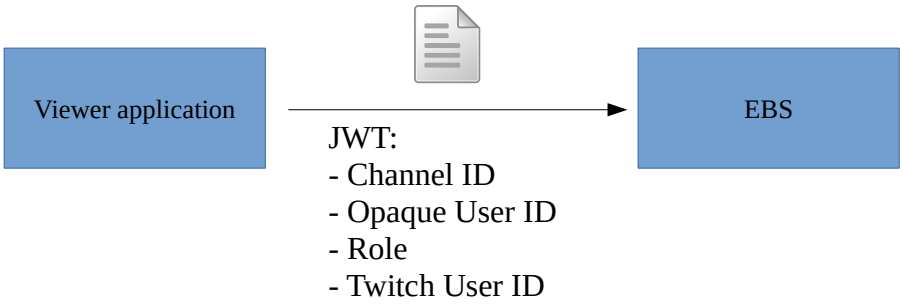
Al seleccionar "Use Extension tutorial example" como origen del código, nos insta a instalar Node.js, un servidor de aplicaciones escrito en Javascript.

Llevamos a cabo la instalación de la versión 13.2. Esta instalación conlleva, en caso de no tenerlos instalados, la instalación adicional de Python 2.7.17, Chocolatey, un gestor de paquetes para Windows, y Virtual Studio 2017. Tras finalizar la instalación es necesario reiniciar.

Una vez seleccionado el ejemplo "Getting started" y haciendo click en "Next" ya hemos finalizado la creación.

5.2.2 Autorización / Autenticación del cliente

El *Frontend* es provisto por la API de Twitch de un testigo JSON *Web Token* o JWT codificado en Base64 y firmado con un secreto compartido entre Twitch y el EBS, por lo que el usuario no puede saber el contenido del testigo ni modificarlo puesto que no conoce el secreto.



Código JavaScript del front end

La aplicación cliente necesita referenciar la API javascript de Twitch en el código del frontend. Esto se hace añadiendo la siguiente linea a la cabecera del código HTML

```
<script src="https://extension-files.twitch.tv/helper/v1/twitch-ext.min.js" />
```

El archivo `twitch-ext.min.js` contiene todas las funciones necesarias para **autorizar la comunicación** entre la extensión y el EBS, y se lleva a cabo **de forma transparente** al programador de la extensión.

Adicionalmente, el código HTML inserta el código de script <script src="viewer.js" type="text/javascript" />, que es el que hace las llamadas a las funciones de `twitch-ext.min.js` e implementa el comportamiento de la interfaz de usuario.

Para la autorización del *front end*, simplemente hay que implementar el código de la función de *callback* `twitch.onAuthorized,` dentro del archivo `viewer.js`, tal y como podemos ver en el ejemplo que creado:

```
twitch.onAuthorized(function (auth) {
  // almacenamos nuestras credenciales
  token = auth.token; // Este es el token JWT encriptado
  tuid = auth.userId; // Este es el Id de usuario

  // Hacemos lo que queramos con nuestra interfaz de usuario
  ...

  setAuth(token); // configura las cabeceras de las peticiones para
insertar el token
  $.ajax(request); // Hacemos la petición al EBS
});
```

Una vez autorizados a comunicarnos con el EBS, creamos la solicitud y la enviamos mediante Ajax pasándole un **objeto configurable**[12] *request* como el siguiente:

```
request = {
    type: type,
    url: location.protocol + '//localhost:8081/color/' + method,
    success: updateBlock,
    error: logError
  };
```

Este objeto proporciona la información siguiente:

- Type: tipo de solicitud (POST o GET)
- URL: la URL de destinataria de la solicitud
- success: función de *callback* a ejecutar cuando la solicitud tenga éxito.
- error: función de *callback* a ejecutar cuando la solicitud falle.

Finalmente, vemos cómo en el ejemplo se envía el *token* recibido en la función twitch.onAuthorized dentro de las cabeceras de la solicitud:

```
request.headers = { 'Authorization': 'Bearer ' + token };
```

Debido a la falta de tiempo y conocimientos, se modifica un ejemplo de extensión que interactúa con el chat de Twitch, que es una parte primordial para el funcionamiento de nuestra extensión. El código del que partimos se obtiene de https://github.com/TwitchDev/chat-samples/tree/master/python

5.3 Desarrollo del Extension Backend Service

5.3.1 Preparación del entorno de desarrollo

Se instala el *framework* de aplicaciones web Django, el cual nos permitirá alojar nuestro *back end* escrito en Python. El motivo para elegir Django es principalmente porque nos proporciona una serie de módulos ya creados para facilitar el desarrollo desde cero de una aplicación web, ya que, como defienden sus creadores, "no hay que reinventar la rueda". Además, está escrito en Python, por lo que nos proporciona toda la flexibilidad que nos ofrece este lenguaje de programación.

Django adopta una arquitectura MVC (*Model-View-Controller*), por lo que separa la lógica de la aplicación de la presentación y el modelo. Al recibir una petición, el servidor analiza la URL mediante la clase *urlresolver* y redirige la petición a la función (o vista) correspondiente.

Se instalan los paquetes de Python necesarios para el desarrollo y ejecución del EBS utilizando el gestor de paquetes pip.

```
pip3 install django, djangorestframework,
djangorestframework-jwt
```

12 Fuente: https://api.jquery.com/jQuery.ajax/

5.3.2 Creación del proyectoDjango

Una vez instalado el *framework*, se procede a crear el proyecto del EBS ejecutando el comando siguiente dentro del directorio donde lo queramos crear:

```
django-admin startproject EBS
```

La ejecución del comando anterior nos crea una estructura de directorios y archivos de configuración necesarios. Comprobamos que el proyecto se ha creado correctamente ejecutando:

```
py -3 manage.py runserver
```

Se hace la llamada con py -3 para forzar la utilización de la versión de Python 3.6 ya que la siguiente orden se ejecuta con la versión 2.7, con la que la sintaxis de manage.py no es compatible:

```
python manage.py runserver
```

Si todo es correcto, se debería mostrar el siguiente mensaje en el intérprete de comandos:

```
PS G:\Mis documentos\UOC\TFG\django_projects\ebs> py -3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you a
pply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
December 02, 2019 - 17:39:19
Django version 3.0, using settings 'ebs.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figura 41: Respuesta de puesta en marcha del servidor web de Django

A partir de este momento, tenemos el servidor de Django escuchando peticiones en el puerto 8000.

Para empezar a trabajar en el EBS, partimos del tutorial que se encuentra en la web oficial de Django (https://docs.djangoproject.com/en/2.2/intro/tutorial01/).

Junto al archivo manage.py, se crea la estructura de directorios y archivos de nuestra aplicación, a la que llamaremos predictor:

```
py -3 manage.py startapp predictor
```

Se crean los siguientes elementos:

```
predictor/
  migrations:
  admin.py
  apps.py
  models.py
  tests.py
  views.py
  urls.py
  __init__.py
```

De esos archivos, los más relevantes (de momento) son los archivos views.py y urls.py.

El archivo ***views*** es el que contendrá las implementaciones de los controladores para las solicitudes recibidas. En el ejemplo del tutorial se crea el método *index(request)* que devuelve un mensaje de texto al cliente:

```
from django.shortcuts import render
  from django.http import HttpResponse

  # Create your views here.
  def index(request):
    return HttpResponse("Hello, world. You're at the predictor
                                    index.")
```

En en caso de nuestra aplicación, el contenido de la respuesta *HttpResponse*, dependiendo de la solicitud, deberá ser una lista de tuplas, un valor numérico, etc.

Ahora solamente hay que enlazar esa función con la URL de la correspondiente solicitud, editando el archivo *urls.py* y referenciándola en el patrón que corresponda:

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Dentro de este archivo se define la lista *urlpatterns*, la cual contiene los patrones que necesitamos que la aplicación reconozca y que utilizaremos como interfaz REST. En el ejemplo anterior se configura el patron 'vacío' que llama al método *views.index* definido anteriormente. Lo único que queda por hacer es enlazar estos patrones, propios de la aplicación *predictor*, con los patrones del proyecto EBS, editando también el archivo *urls.py* del directorio del proyecto *ebs/urls.py*:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('predictor/', include('predictor.urls')),
    path('admin/', admin.site.urls),
]
```

Se añade a la lista de patrones la linea `path('predictor/', include('predictor.urls'))`, que sirve para indicarle al servidor que todo lo que siga al patrón 'predictor/' se gestiona desde el archivo predictor/urls, mediante la función `include`.

A partir de este ejemplo, podemos añadir tantos patrones como necesitemos solo con crear las respectivas funciones y referenciándolas en el archivo *urls* correspondiente.

La primera acción que realiza el EBS es autenticar el cliente mediante el token JWS que éste nos envía.

El primer problema que encontramos es que la cabecera *Authorization* no tiene valor cuando la leemos desde nuestro EBS. Por algún motivo el servidor de Django no está guardando o no está recibiendo el valor del *token*. Al comprobar el log de Django vemos que el *front end* está enviando solicitudes HTTP OPTIONS en lugar de GET o POST.

Tras una larga búsqueda averiguamos que la causa del problema es la política *Same-Origin* que se implementa en la mayoría de exploradores de Internet mediante mecanismos **CORS** (Cross-Origin Resource Sharing), que requieren cabeceras HTTP adicionales para permitir que un user agent obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece. Un agente crea una petición HTTP de origen cruzado cuando solicita un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que lo generó (*https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS*).

Así pues, estamos incumpliendo dos políticas de seguridad:

1. Se envían solicitudes desde un dominio o puerto diferente, ya que el front end envía las solicitudes desde localhost:49220, que es diferente al puerto 8000 del servidor de localhost donde se aloja el backend.

2. Se utiliza la cabecera 'Authorization', que no es una cabecera permitida por defecto para solicitudes de sitio cruzado.

Por ese motivo, cuando se envía esa solicitud OPTIONS llamada ***Preflighted*** o de verificación y el servidor no indica en la respuesta que acepta ese origen y esa cabecera, cesa en el envío de la verdadera solicitud GET o POST.

Para solucionarlo es necesario especificar en el lado del servidor que se aceptan los orígenes de cualquier dominio y adicionalmente que también se permite la cabecera 'Authorization'.

Dentro de la clase del controlador de la vista de autorización que responde a las solicitudes de la URL *http://localhost:8000/predictor/*, se añade al método manejador de las peticiones OPTIONS, options(), las siguientes lineas para

añadir a la respuesta las cabeceras Access-Control-Allow-Origin y Access-Control-Allow-Headers:

```
    def options(self, request):
        response = Response()
        response['Access-Control-Allow-Origin']='*'
        response['Access-Control-Allow-Headers'] =
'Authorization'

    return response
```

Finalmente, la petición Ajax funciona como debería y somos capaces de visualizar el token enviado en las peticiones del cliente. Realizamos pruebas mediante GET y POST y obtenemos la siguiente salida del servidor por la consola, donde imprimimos desde nuestra aplicación el token recibido, enmarcado en verde:

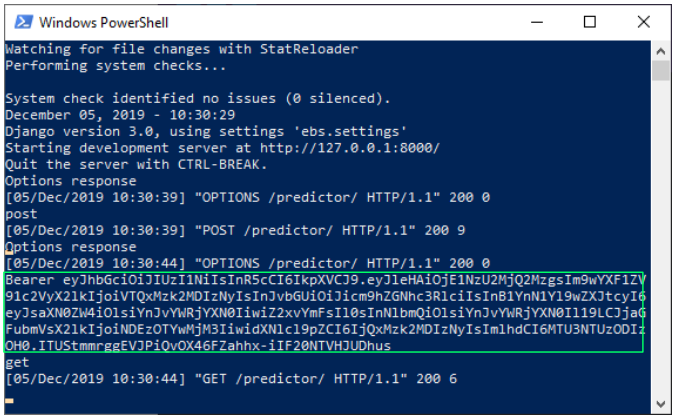


Figura 42: Salida del servidor tras solicitudes GET y POST

5.3.3 Validación del token JWT

Se usa como referencia el proyecto https://github.com/HearthSim/twitch-hdt-ebs/tree/master/twitch_hdt_ebs para hacernos una idea de como validar el token JWT.

El *token* está codificado en Base64 y firmado usando el algoritmo HMAC-SHA256. Cuando lo decodificamos obtenemos lo siguiente:

Tabla 17: Estructura de un token JWT

JWT Codificado (Base64)
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NzU4MTQ5NzUsIm9wYXF1ZV91c2VyX2lkIjoiVTQxMzk2MDIzNyIsInJvbGUiOiJicm9hZGNhc3RlciIsInB1YnN1Yl9wZXJtcyI6eyJsaXN0ZW4iOlsiYnJvYWRjYXN0IiwiZ2xvYmFsIl0sInNlbmQiOlsiYnJvYWRjYXN0Il19LCJjaGFubmVsX2lkIjoiNDEzOTYwMjM3IiwidXNlcl9pZCI6IjQxMzk2MDIzNyIsImlhdCI6MTU3NTcyODU3NX0.MZO1yHxyR_Zk6YfDXdA5c-eWqY7qnUE3-S4bvA9yZ78
JWT Descodificado

Cabecera	Contenido (*Payload*)	HMAC-SHA256

{
"alg": "HS256",
"typ": "JWT"
}
	{
	"exp": 1575814975,
	"opaque_user_id": "U413960237",
	"role": "broadcaster",
	"pubsub_perms": {
	"listen": [
	"broadcast",
	"global"
	],
	"send": [
	"broadcast"
	]
	},
	"channel_id": "413960237",
	"user_id": "413960237",
	"iat": 1575728575
	}
		MZO1yHxyR_Zk6YfDXdA5c-eWqY7qnUE3-S4bvA9yZ78

El *framework* django-rest nos permite declarar en nuestras clases manejadoras del EBS qué clases queremos utilizar para autenticar los usuarios que realizan las peticiones. Esto se puede hacer a nivel global de aplicación modificando el archivo settings.py dentro del directorio del proyecto *ebs/* o bien a nivel de clase, dentro de cada clase view.
A nivel global (settings.py):

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (

'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}
```

A nivel de clase (views.py):

```
authentication_classes = [SessionAuthentication,
BasicAuthentication]
```

No obstante, estas clases buscan por defecto en las bases de datos de usuario que se le indiquen al *framework*, mientras que nosotros no queremos recuperar el usuario sino simplemente verificarlo. Para ello, debemos crear nuestra clase personalizada extendiendo la clase rest_framework.authentication.BaseAuthorization.

Tal y como se explica en la documentación de django-rest[13], extendemos dicha clase y definimos el método `authenticate()` que validará el *token*. A grandes rasgos, el pseudocódigo del método es el siguiente:

13 https://www.django-rest-framework.org/api-guide/authentication/

```
function authenticate(request): user_id
    token := get_header( request, 'Authorization');
    secret := get_environment_var( 'SECRET' );
    user_id := get_environment_var( 'USER_ID' );

    decoded_secret := decode( secret );
    if token_is_valid( token, secret ) then
        return user_id;
    else
        error( 'Token not valid' );
        return NULL;
```

5.3.1 Conexión con el servidor IRC de Twitch

Para poder leer los mensajes del chat desde el EBS es necesario conectarse con el servidor de IRC de Twitch, que escucha el puerto 6667 en irc.chat.twitch.tv, e identificarse con un token Oauth (Open Authorization). Oauth es un protocolo abierto, que permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web (Wikipedia).

Tal y como nos indica la documentación de Twitch , la solicitud se puede realizar por dos vias: a través de la API de Twitch o mediante un formulario web accesible en https://twitchapps.com/tmi/.

Si optamos por la API de Twitch v5 debemos realizar la siguiente petición:

```
GET https://id.twitch.tv/oauth2/authorize?
client_id=<your client ID>&
redirect_uri=<your registered redirect URI>&
response_type=code&
scope=<space-separated list of scopes>
```

Tras enviar la solicitud, podemos encontrar el código Oauth dentro de la URL de redirección:

```
https://<your registered redirect URI>#access_token=<an access
token>
```

No obstante, se opta por la segunda opción debido a su simplicidad y rapidez ya que simplemente es necesario visitar la URL indicada y aceptar la solicitud.

Antes de realizar la conexión debemos realizar un paso previo, que consiste en obtener el nombre del canal al que queremos conectaros, ya que hasta ahora solamente conocemos el channel_id contenido en el token JWT, y es un dato imprescindible para acceder a la sala de chat correspondiente.

Mediante el módulo requests de Python, se realiza la petición a la API de Twitch a través de la URL correspondiente:

```
url = 'https://api.twitch.tv/kraken/channels/’ + channel_id
headers = {

  'Client-ID': client_id,
  'Accept': 'application/vnd.twitchtv.v5+json'

}
r = requests.get(url, headers=headers).json()
```

El nombre del canal lo encontramos bajo la clave ‘name’ del objeto JSON devuelto.

La conexión con el servidor de IRC de Twitch se realiza mediante un *socket* de Python conectado con el servidor irc.chat.twitch.tv en el puerto 6667. Los pasos a seguir una vez conectados al servidor son los siguientes:

Para autentificar nuestra identidad en el servidor, se envían los siguientes mensajes:

PASS oauth:<OAuth token>
NICK <nombre_usuario>

Donde <Oauth token> es el *token* de autorización obtenido previamente y <nombre_usuario> debería ser el mismo nombre de la cuenta autorizada por el *token*.

Para unirnos a un canal (el del chat de la cuenta del cliente) se envía el mensaje siguiente:

JOIN #<nombre_canal>

Donde <nombre_canal> será el valor ‘name’ obtenido en la solicitud del punto anterior.

Se leen los mensajes mediante la operación recv() del socket:

```
response = self.irc.recv(2040).decode("UTF-8")
```

Al conectarnos por primera vez al servidor y unirnos al canal, recibimos del servidor una serie de mensajes con información sobre el canal al que nos hemos conectado:

```
:username!username@username.tmi.twitch.tv JOIN #channel
:username .tmi.twitch.tv 353 username = #channel :
username
:username .tmi.twitch.tv 366 username #channel :End of
/NAMES list
```

Puesto que no nos interesa procesar esta información, se filtran los mensajes recibidos para capturar solamente los correspondientes a intervenciones de usuario, que tienen el siguiente formato:

:<user>!<user>@<user>.tmi.twitch.tv PRIVMSG #<channel> :message

Los mensajes de usuario contienen la palabra clave PRIVMSG. Así pues, se seleccionan solamente éstos mediante una simple expresión regular y se extrae el usuario y mensaje con sendas expresiones regulares:

```
    if re.search('PRIVMSG',message):
    username = re.search(":(.*)!",message).group(1)
    text = re.search(username + '\s:(.*)',
message).group(1)
```

Con el usuario y el texto del mensaje obtenidos, solamente nos queda utilizar nuestro clasificador de polaridad para obtener la predicción del texto y enviarla de vuelta al cliente. Para ello, necesitamos ejecutar otra instancia separada de Django para servir las solicitudes.

5.4 Desarrollo del servicio de predicciones

El desarrollo del servidor de predicciones sigue el mismo proceso que el del EBS, slavo que en lugar de escuchar el puerto 8000 se ejecuta la aplicación para que escuche el puerto 8001 a través de la URL http://localhost:8001/predict.

El código se puede encontrar en la carpeta "código/Extensión/Django Project/Prediction_server/Prediction_model" dentro del archivo comprimido que se adjunta a la memoria.

6 Conclusiones

El análisis de sentimiento no es una tarea sencilla incluso para los seres humanos, siendo a menudo más sencillo interpretar la polaridad de un texto claramente positivo o negativo que la de un texto neutro o no opinable.

A simple vista, los resultados obtenidos pueden parecer malos resultados aunque, en nuestra opinión, el modelo conseguido tiene una exactitud bastante aceptable ya que en muchos casos los errores se deben a que el matiz que diferencia un texto de una categoría de otra es muy sutil y al final se elige la clase con la confianza más alta.

No obstante, surge la cuestión de si ese bajo rendimiento realmente se debe a una mala configuración de los hiperparámetros, una mala elección de la arquitectura o bien está causado por un problema intrínseco del lenguaje o de la forma en que están etiquetados los textos, ya que incluso a un servidor le es complicado discernir la polaridad de los textos evaluados incluidos en el corpus debido a diferencias demasiado sutiles entre textos de diferentes categorías. La polaridad de un texto está ligada a su interlocutor y el contexto, y en determinadas ocasiones varía en función de la situación de éste.

6.2 Sobre la arquitectura elegida

La arquitectura elegida para la predicción con tres niveles de polaridad (P, N y Otras) es la arquitectura LSTM de una capa elaborado con un vocabulario lematizado de 10000 palabras, ya que es con la que mejor media de exactitud hemos obtenido, de un 76,907, seguida por el modelo MLP con 150 neuronas y vocabulario lematizado también de 10000 palabras.

Se elige el modelo en base a su exactitud media y no la exactitud máxima ya que la exactitud máxima se consigue con conjuntos de datos concretos, mientras que la media nos parece un indicador de regularidad más fiable.

Respecto a la arquitectura MLP, tiene el problema de que la representación de características como vectores bag of words requiere cantidades ingentes de memoria para vectorizar los conjuntos de entrenamiento y prueba. En el caso del conjunto de entrenamiento usando, que es relativamente pequeño, de 60798 textos y usando un vocabulario de 10000 palabras se obtiene un vector de 60798x10000 = 607.980.000 números enteros que ocupan más de 2 GB en disco, por lo que escala bastante mal al aumentar el vocabulario o el número de textos. En comparación, la arquitectura LSTM requiere muy poco espacio ya que los textos se codifican en secuencias de 40 índices, por lo que el tamaño del corpus es de 60798x40 mas el espacio que ocupa el diccionario de mapeo palabra:índice.

En cualquier caso, se considera que ambos modelos son perfectamente utilizables ya que la diferencia de rendimiento es inapreciable a la práctica.

6.3 Sobre la metodología utilizada

Un aspecto potencialmente problemático que hemos detectado durante el desarrollo tiene que ver con la fiabilidad de lematización a la hora de procesar palabras homógrafas (aquellas que se escriben igual). Como ejemplo, la palabra ‘siento’ puede referirse a la primera persona del singular del indicativo del verbo ‘sentir’ o ‘sentirse’, pero también del verbo ‘sentar’ o ‘sentarse’; la primera es una palabra que expresa sentimiento mientras que la segunda no. Esto es comprensible dada la falta de información sobre el contexto y afortunadamente los casos en los que se produce esta ambigüedad son minoría.

También se hace hincapié en la dificultad que supone durante la fase de preprocesamiento de los textos la corrección de errores gramaticales intencionados o accidentales que pueden ocasionar errores en la predicción. En el texto 4 de la lista mostrada anteriormente podemos ver un ejemplo en el que se escribe la palabra ‘Noooooos’ en lugar de ‘buenos’. Nuestro algoritmo la preprocesa como ‘Nos’ tras la eliminación de caracteres repetidos y la obvia al hallarse en la lista de palabras vacías, perdiendo la frase significado. Otro ejemplo real es la frase ‘Muy interresante las comparaciones recibidas’ donde ‘interresante’ está mal escrita y no se tiene en cuenta esa palabra al tokenizar la frase ya que no existe en el vocabulario. Así pues, creemos que la incorporación de algún tipo de corrector ortográfico o diccionario sería bastante útil a la hora de evitar este tipo de errores.

Acerca de la planificación, se ha intentado seguirla al pie de la letra tanto como ha sido posible, pero en ocasiones ha surgido algún obstáculo que ha costado superar debido a la falta de conocimiento y experiencia y ha obligado a revisar la planificación para ampliar el plazo de tiempo asignado inicialmente a alguna actividad, tal y como se ha indicado en los informes realizados a lo largo del proyecto.

A pesar de todo, la planificación inicial ha ayudado mucho a la hora de establecerse metas parciales y a no perderse ante tal cantidad de trabajo.

6.4 Objetivos no alcanzados

A la fecha de esta memoria no ha sido posible finalizar el desarrollo de la extensión y el despliegue del modelo que tenía como objetivo demostrar la aplicación práctica en un escenario real puesto que se ha dedicado más tiempo del previsto en la creación del modelo y la realización de pruebas, lo cual ha relegado el desarrollo de la extensión a un segundo plano.

6.5 Lineas de trabajo futuras

En el futuro, se pretende solventar los errores comentados en el apartado anterior incorporando un corrector ortográfico para corregir los errores ortográficos, así

como probar la arquitectura LSTM bidireccional, que no hemos podido probar por falta de tiempo así como más configuraciones para la red convolucional o combinación de varias arquitecturas tales como CNN + LSTM.

7. Glosario

- JWT: JSON Web Token. Formato de testigo de autenticación de usuario para el acceso a recursos remotos.
- JSON: Javascript Object Notation.
- CNN: Convolutional Neural Network
- IG: Information Gain.
- MLP: Multi-Layer Perceptron
- EBS: Extension Backend Service
- LSTM: Long Short Term Memory:
- Dropout: técnica de regularización consistente en anular un determinado número de neuronas en cada iteración de entrenamiento.
- RD: Reducción de Dimensionalidad
- RDS: Reducción de Dimensionalidad por Selección
- RDE: Reducción de Dimensionalidad por Extracción
- RNN: Recurrent Neural Network
- PMI: Pointwise Mutual Information
- POS: Part of Speech
- NB: Naïve Bayes
- SVM: Support Vectors Machine
- KNN: K-Nearest Neighbours
- TF-IDF: Term Frequency – Inverse Document Frequency
- CORS: Cross Origin Resource Sharing
- Ajax: Asynchronous JavaScript and XML
- IRC: Internet Relay Chat
- HTTP: Hypertext Transfer Protocol

8. Bibliografía

[1] Medium. (2019). A Practitioner's Guide to Natural Language Processing (Part I) — Processing & Understanding Text. [online] Available at: https://towardsdatascience.com/a-practitioners-guide-to-natural-language-processing-part-i-processing-understanding-text-9f4abfd13e72 [Accedido 10 Oct. 2019].

[2] Twitch Developers. (2019). Required Technical Background. [online] Available at: https://dev.twitch.tv/docs/extensions/required-technical-background/ [Accedido 10 Oct. 2019].

[3] B. Liu, Handbook Chapter: Sentiment Analysis and Subjectivity. Handbook of Natural Language Processing, Handbook of Natural Language Processing, Marcel Dekker, Inc. New York, NY, USA, 2009. [Consultado: 17/10/2019]

[4] Mika V. Mäntylä et al, The evolution of sentiment analysis—A review of research topics, venues, and top cited papers, Computer Science Review, Volume 27, 2018, pp. 16-32 [Consultado: 17/10/2019]

[5] B. Pang, L. Lee, S. Vaithyanathan, Thumbs up?: sentiment classification using machine learning techniques, in: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing-Volume 10, 2002, pp. 79–86. [Consultado: 17/10/2019]

[6] B. Pang, L. Lee, Opinion mining and sentiment analysis, Found. Trends Inf. Retr. 2 (1–2) (2008) 1–135. [Consultado: 18/10/2019]

[7] R. Feldman, Techniques and applications for sentiment analysis, Commun. ACM 56 (4) (2013) 82–89. [Consultado: 18/10/2019]

[8] Musto, Cataldo & Semeraro, Giovanni & Polignano, Marco. (2014). A comparison of lexicon-based approaches for sentiment analysis of microblog. CEUR Workshop Proceedings. 1314. 59-68. [Consultado: 18/10/2019]

[9] A. P. Jain and P. Dandannavar, "Application of machine learning techniques to sentiment analysis," 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Bangalore, 2016, pp. 628-632. [Consultado: 18/10/2019]

[10] Moidigital.ac.uk. (2019). User Guide. [online] Available at: http://www.moidigital.ac.uk/user-guide/ [Consultado: 18/10/2019]

[11] Fabrizio Sebastiani. Machine learning in automated text categorization. ACM Computing Surveys,
34(1):1–47, 2002. [Consultado: 19/10/2019]

[12] Haryanto, Ardy & Mawardi, Edy & Muljono,. (2018). Influence of Word Normalization and Chi-Squared Feature Selection on Support Vector Machine (SVM) Text Classification.[Consultado: 19/10/2019]

[13] Pu Han;Si Shen;Dongbo Wang;Yanyun Liu (2012). The influence of word normalization in English document clustering. IEEE International Conference on Computer Science and Automation Engineering (CSAE) , 2012 IEEE International Conference on. 2:116-120 May, 2012 [Consultado: 19/10/2019]

[14] K M M, Rajashekharaiah & Chikkalli, Sunil & Kumbar, Prateek & Babu, P & Supervisor, Research. (2018). Unified Framework of Dimensionality Reduction and Text Categorisation. [Consultado: 20/10/2019]

[15] M. Mhatre, D. Phondekar, P. Kadam, A. Chawathe, K. Ghag, "Dimensionality reduction for sentiment analysis using pre-processing techniques", 2017 International Conference on Computing Methodologies and Communication (ICCMC), pp. 16-21, 2017. [Consultado: 20/10/2019]

[16] Emma Haddi, Xiaohui Liu, Yong Shi, The Role of Text Pre-processing in Sentiment Analysis,Procedia Computer Science, Volume 17, 2013, Pages 26-32, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2013.05.005.(http://www.sciencedirect.com/science/article/pii/S1877050913001385) [Consultado: 20/10/2019]

[17] Es.wikipedia.org. (2019). Punto de información mutua. [online] Available at: https://es.wikipedia.org/wiki/Punto_de_informaci%C3%B3n_mutua [Consultado: 23/10/2019].

[18] Kyoungok Kim, An improved semi-supervised dimensionality reduction using feature weighting: Application to sentiment analysis, Expert Systems with Applications, Volume 109, 2018, pp. 49-65, [Consultado: 21/10/2019]

[19] Murillo, E. C., y G. M. Raventós. 2016. Evaluación de Modelos de Representación del Texto con Vectores de Dimensión Reducida para Análisis de Sentimiento. En Proceedings of TASS 2016: Workshop on Sentiment Analysis at SEPLN co-located with 31st SEPLN Conference, páginas 23-28. [Consultado: 21/10/2019]

[20] Yu, L., Ye, J., & Liu, H. (2007). Dimensionality Reduction for Data Mining-Techniques, Applications and Trends. In Proc. SIAM Int. Conf. Data Min. Proc.. [Consultado: 21/10/2019]

[21] En.wikipedia.org. (2019). Information gain in decision trees. [online] Available at: https://en.wikipedia.org/wiki/Information_gain_in_decision_trees [Consultado: 24/10/2019]

[22] K. Dave, S. Lawrence, and D. M. Pennock, “Mining the peanut gallery: Opinion extraction and
semantic classification of product reviews,”Proceedings of WWW, 2003, pp. 519–528. [Consultado: 24/10/2019]

[23] A.Krouska, C.Troussas, M.Virvou (2016), The effect of preprocessing techniques on Twitter sentiment analysis 7th International Conference on Information, Intelligence, Systems & Applications (IISA) [Consultado: 25/10/2019]

[24] G. Sidorov, S. Miranda-Jiménez, F. Viveros-Jiménez, .., and J. Gordon,“Empirical study of machine learning based approach for opinion mining in tweets”, 11th Mexican International Conference on Artificial Intelligence, MICAI, 2012.[Consultado: 25/10/2019]

[25] K. Jain and S. Kaushal, "A Comparative Study of Machine Learning and Deep Learning Techniques for Sentiment Analysis," 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2018, pp. 483-487.[Consultado: 25/10/2019]

[26] S. Vohra and J. Teraiya. A comparative study of sentiment analysis techniques. Journal JIKRCE, 2(2):313--317, 2013 [Consultado: 23/10/2019]

[27] M. Hu and B. Liu, "Mining and summarizing customer reviews," Proceedings of the tenth ACM international conference on Knowledge discovery and data mining, Seattle, 2004, pp. 168-177.[Consultado: 23/10/2019]

[28] Khan MT(1), Khalid S(1), Durrani M(2), Ali A(2), Inayat I(3), Khan KH(4). Sentiment analysis and the complex natural language. Complex Adaptive Systems Modeling. 4(1). [Consultado: 25/10/2019]

[29] Turney, P. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In Proceedings of the Association for Computational Linguistics (2002), 417–424.[Consultado: 27/10/2019]

[30] Wiebe, J., & Riloff, E. (2005, February). Creating subjective and objective sentence classifiers from unannotated texts. In International conference on intelligent text processing and computational linguistics (pp. 486-497). Springer, Berlin, Heidelberg. [Consultado: 27/10/2019]

[31] Dos Santos, C., & Gatti, M. (2014, August). Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers (pp. 69-78) [Consultado: 28/10/2019]

[32] Pasupa, K., & Ayutthaya, T. S. N. (2019). Thai Sentiment Analysis with Deep Learning Techniques: A Comparative Study based on Word Embedding, POS-tag, and Sentic Features. Sustainable Cities and Society. [Consultado: 28/10/2019]

[33] Géron, A. (n.d.). Hands-on machine learning with Scikit-Learn and TensorFlow.

1st ed. O'Reilly, pp.379-407. [Consultado: 28/10/2019]

[34 Go, A., Bhayani, R., & Huang, L. (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, *1*(12), 2009. [Consultado: 12/12/19]

[35] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *preprint arXiv:1609.04747*. [Consultado: 12/12/19]