UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Procesamiento del lenguaje natural para el análisis de la crispación política en España

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Garrido Martínez

Tutor: Lluís F. Hurtado Oliver

Co-tutor: Ferran Pla Santamaria

Curso: 2020/2021

Resumen

Analizar el clima de crispación política de una sociedad es muy importante para estudiar su comportamiento y sus reacciones. Cuando hablamos de crispación política normalmente la asociamos a ciudadanos anónimos, noticias falsas y redes sociales. Sin embargo, muchas veces los profesionales de la política contribuyen de forma significativa a esa crispación.

En este Trabajo Final de Grado se plantea el análisis del clima político en España, para ello se hará un seguimiento de las intervenciones de los diputados en el congreso en los últimos años. Se utilizarán técnicas y herramientas de procesamiento del lenguaje natural para analizar la polaridad de los discursos.

Palabras clave: Procesamiento de Lenguaje Natural, Análisis de sentimientos, Word Embedding, Deep Learning, Política.

Abstract

Analyzing the climate of political tension in a society is very important to study its behavior and reactions. When we talk about political tension, we usually associate it with anonymous citizens, fake news and social networks. However, many times the professionals of politics significantly to that tension.

In this Final Degree Project the analysis of the political climate in Spain is proposed, for this purpose, the interventions of the deputies in the congress in recent years will be monitored. Natural language processing techniques and tools will be used to analyze the polarity of the speeches.

Keywords: Natural Language Processing, Sentiment Analysis, Word Embedding, Deep Learning, Politics.

Tabla de contenidos

1. Introducción

En los últimos años el análisis de sentimiento es un problema ampliamente estudiado en el campo del procesamiento del lenguaje natural y el aprendizaje automático. Cada vez más, existe una gran cantidad de opiniones en diferentes medios tales como redes sociales, blogs, páginas de comercio electrónico, etc. Estas opiniones generalmente son comentarios escritos en lenguaje natural que pueden ir acompañados por algún tipo de valoración del usuario como, por ejemplo, si está satisfecho o no con lo que comenta, una valoración numérica en forma de 1 a 5 estrellas, etc.

Hoy en día, existen múltiples técnicas basadas en *machine learning* para abordar de manera automática la tarea del análisis de sentimientos. Estas técnicas también pueden hacer uso de diferentes recursos que ayuden a resolver el problema como diccionarios de polaridad, diccionarios de palabras malsonantes, etc. Las aproximaciones que mejores resultados obtienen son las que se basan en aprendizaje automático clásico, o más recientemente, técnicas de aprendizaje profundo basadas en redes neuronales artificiales. Estos modelos necesitan grandes cantidades de textos para construir el sistema de análisis de sentimientos.

Sin embargo, no existen muchos trabajos que apliquen este tipo de análisis a los discursos que realizan los políticos durante sus intervenciones en los parlamentos y menos los que lo aplican para el español. Entre los trabajos más relevante se puede encontrar el realizado por Rheault et al. (2016), donde se midió la emoción en los debates parlamentarios de la cámara de los comunes de Reino Unido durante el siglo XX. [11].

2. Objetivos

El objetivo principal del trabajo es determinar la polaridad que existe en el Congreso de los Diputados español dado que, hasta donde sabemos, no existe ningún trabajo anterior que realice esta tarea.

Para ello, en primer lugar, se debe realizar un proceso de recopilación de las actas de las sesiones plenarias del congreso. A continuación, se analizarán usando un modelo de *Deep Learning* para extraer la polaridad de la opinión que se expresa. Además, se realizará un modelo de *Word Embedding* con las actas para calcular las similitudes semánticas entre distintos conceptos como, por ejemplo, siglas y nombres de partidos políticos y políticos concretos de estos partidos.
Además de ello, se compararán las palabras obtenidas por nuestros *Word embeddings* con la nube de palabras obtenida en [23].

3. Estado del arte

En este capítulo se introducen conceptos básicos de Procesamiento de Lenguaje Natural, Análisis de sentimientos y *Deep Learning*. Además, se presentan algunos trabajos que se han realizado hasta ahora sobre análisis de sentimientos en política.

Dado que vamos a realizar la tarea del análisis de sentimientos mediante el análisis del discurso a través del texto de las actas, necesitamos del uso de herramientas pertenecientes al Procesamiento del Lenguaje Natural, un área de investigación y aplicación que explora como las computadoras pueden ser usadas para entender y manipular lenguaje oral o escrito para hacer cosas útiles [30].

Esta área de la informática ha tenido un gran desarrollo en los últimos años debido al acceso a grandes cantidades de datos a través de internet (p.ej. redes sociales o digitalización de textos) y al uso de técnicas de inteligencia artificial de procesamiento de lenguaje natural para gestionar estos datos como el análisis de sentimientos [2].

3.1 Análisis de sentimiento

El análisis de sentimiento es el estudio computacional de las opiniones, apreciaciones, actitudes y emociones de la gente hacia entidades, individuales, problemas, eventos, temas y sus atributos [29].

El análisis de sentimiento usado en la política tiene antecedentes por ejemplo en [12] y [13] donde se usa el análisis de sentimiento para entender que es lo que piensan los votantes. Otros trabajos, han experimentado clasificando medios de comunicación en internet que tienden hacia una parte del espectro político u otro. [14]

Estos y otros trabajos ya muestran que es difícil la clasificación entre sentimiento positivo y negativo y más en el ámbito de la política donde es habitual el uso de la ironía entre otros recursos del lenguaje.

Otro de los trabajos que ha usado machine learning en el análisis de sentimiento es "*Exploring the political pulse of a country using data science tools*" [22] donde los autores utilizan los *tweets* de las cuentas oficiales de los presidentes de los 5 partidos mayoritarios de España (PSOE, PP, VOX, Podemos y Ciudadanos en ese momento) para realizar el análisis de sentimiento. En primer lugar, realizan *Word Clouds* para los principales partidos y posteriormente analizan la evolución del sentimiento de los partidos en Twitter a lo largo del periodo de Enero 2016 hasta Mayo 2020. Los resultados fueron los siguientes:

Para Podemos, las palabras más repetidas en su Twitter fueron: *derecho* y *gente*. Para el PSOE, fueron: *proyecto*, *socialista*, *derecha* y *futuro*. Para el PP y Ciudadanos, las palabras que comparten son: *Sanchez* y *PSOE*. Las diferencias se atestiguan en que ciudadanos se centra en el problema de la independencia de Cataluña con la palabra *Cataluña* mientras que el PP se centra en *Madrid* donde gobiernan la comunidad autónoma y la ciudad. Por último, Vox también tiene gran incidencia de las palabras *Cataluña* y *Barcelona* además de las palabras *españoles* y *Europa*.

En cuanto al análisis de sentimiento, en este trabajo los autores traducen los *tweets* de los políticos previo al análisis dado que van a hacer uso de VADER (*Valence Aware Dictionary and sEntiment Reasoner*) una librería cuyo uso esta únicamente disponible en inglés y que esta ajustada a contextos de medios sociales. [23]
Los resultados se muestran en la **Figura 0**.

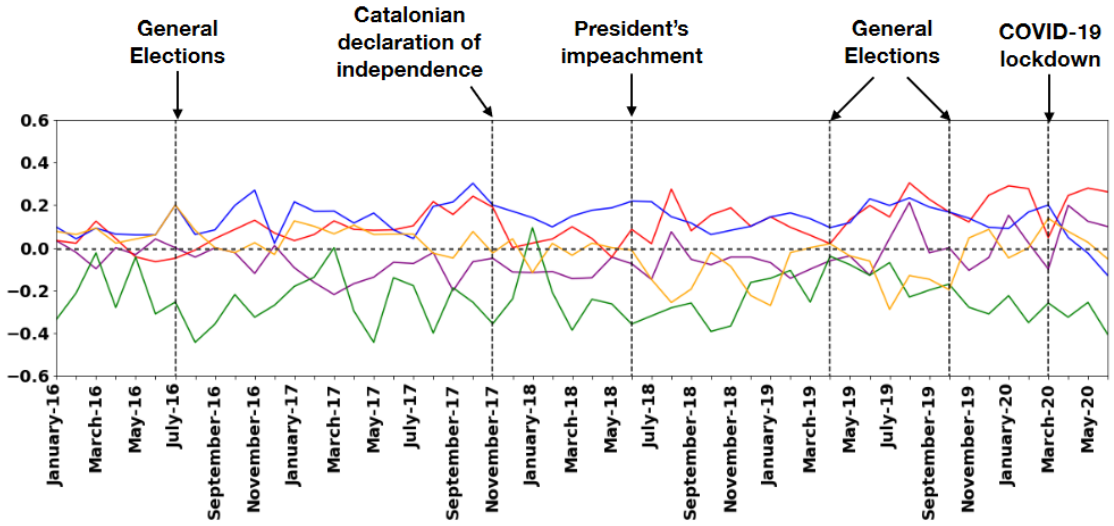


Figura 0: resultado del análisis de sentimientos realizado en Twitter por [23]

En esta gráfica, se pueden ver las tendencias de cada partido, comprobaremos si en nuestro modelo mediante el uso de las actas del congreso, se llegan a conclusiones similares.
Finalmente, el estudio [23] realiza un modelo de *Deep Learning* para la clasificación de *tweets* en el espectro político izquierda-derecha con una precisión del 90% y predecir el origen del *tweet* con una precisión en el rango de 71-75%.

3.1.1 Definición de opinión en análisis de sentimientos

Según la definición dada por Liu et al. [10], un sentimiento (o una opinión) se puede modelar como una quíntupla (**e, a, s, h, t**) donde **e** es el nombre de la entidad de la que se va a referir el sentimiento o la opinión, **a** es un aspecto de **e**, **s** es el sentimiento sobre un aspecto **a** de la entidad **e**, **h** es el origen de la

opinión y **t** es el tiempo en el que se expresó la opinión. En esta definición, el sentimiento **s** puede ser positivo, neutral o negativo o un valor numérico que exprese la fuerza/intensidad del sentimiento (p.ej. 1-5 estrellas) en sitios de críticas como Amazon.

Por ejemplo, para un comentario publicado por un usuario llamado Bob en Amazon:
"Las fotos de la cámara de mi Samsung no son muy buenas pero la duración de batería es genial." El 4 de junio de 2015.

En este ejemplo, hay dos quíntuplas de opinión: (Samsung, foto, negativo, Bob, 4/6/2015) y (Samsung, duración batería, positivo, Bob, 4/6/2015).
Basándonos en esta definición de sentimiento, el objetivo del análisis de sentimientos es encontrar todas las quíntuplas de sentimiento en un documento. Las tareas de análisis de sentimientos tienen que ver con las derivaciones posibles de la definición dada. Por ejemplo, la tarea de clasificación del sentimiento a nivel de documento se centra en la tercera componente de la definición (sentimiento pudiendo ser positivo, neutral o negativo) mientras que se ignoran el resto de los aspectos. La tarea de la extracción fina de opiniones se centra en las 4 primeras componentes de la definición, aunque algunos autores también han incluido en esta variante una variedad mayor de clases a clasificar como por ejemplo más niveles de sentimiento positivo/negativo (muy positivo, positivo, neutral, negativo, muy negativo) o abarcando mayor variedad de sentimiento (enfado, felicidad, ironía…), esto también es llamado análisis de emociones [1].

3.1.2 Niveles de análisis

Además de esto, el análisis de sentimientos habitualmente se puede llevar a cabo a 3 niveles distintos: nivel de documento, nivel de frase y nivel de aspecto o característica [1]. Dependiendo del análisis que se quiera realizar, se pueden encontrar corpus anotados a distintos niveles. Es posible que debido a ello y, con el objetivo de conseguir el máximo número posible de muestras para entrenar los modelos, se utilicen corpus con distintas anotaciones, pero haciendo que todas ellas concuerden entre sí para poder clasificar todos los datos de forma uniforme.

El nivel de documento consiste en clasificar un documento entero dentro de alguna de las clases que se hayan definido en el análisis (habitualmente positivo/negativo). Este enfoque es solo útil si en un mismo documento solo se trata sobre un único tema o producto o algún aspecto de este, en caso de que haya múltiples opiniones, no es aplicable en la práctica.

El nivel de oración consiste en clasificar cada frase de un documento en si expresa una opinión positiva o negativa. Además, a este nivel se puede clasificar también la subjetividad de la frase, es decir, si esta contiene hechos e información o si expresa opiniones y puntos de vista personales.

El nivel de entidad y aspecto (o características) clasifica basándose en que una opinión se aplica sobre un objeto y ésta puede estar dirigida hacia uno (o varios) de sus atributos, pero ese aspecto no afecta de manera determinante a la opinión general sobre el objeto, solo la matiza. Por ejemplo, siguiendo el ejemplo anterior de una opinión sobre un teléfono móvil, si el comentario fuese "Me encanta mi nuevo móvil por que la duración de la batería es genial, aunque las fotos de su cámara no sean muy buenas", se podría clasificar como positivo, aunque el comentario no sea totalmente positivo.

3.2 Deep Learning

Recientemente, en la aproximación al problema del análisis de sentimiento, han emergido modelos de *Deep Learning*, como potentes modelos computacionales que descubren intrincadas representaciones semánticas de textos automáticamente de los datos sin necesidad de hacer ingeniería de características [3].

El *Deep Learning* es la aplicación de redes neuronales artificiales (RNA, en ingles *ANNs: Artificial Neural Networks*) para aprender tareas usando redes de múltiples capas. Esto hace que se pueda explotar mucho más poder de aprendizaje (representación) con respecto a cuando solo podían estar formadas en la práctica por una o dos capas y un conjunto pequeño de datos [1]. En la **Figura 1** se puede observar un ejemplo de RNA con una capa oculta.

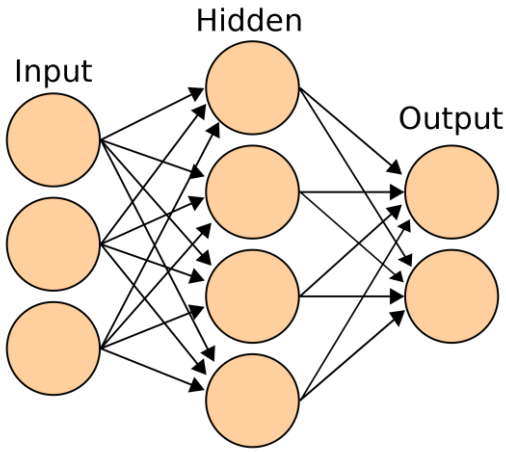


Figura 1: Red Neuronal Artificial con una capa oculta. Extraída de: https://en.wikipedia.org/wiki/Artificial_neural_network#Gallery

Inspiradas por la estructura del cerebro biológico, las redes neuronales consisten en un gran número de unidades de procesamiento de información (llamadas neuronas) organizadas en capas que trabajan al unísono. Las redes neuronales pueden aprender a realizar tareas (p.ej. clasificación) ajustando los pesos de las conexiones entre las neuronas [1].

En resumen, el *Deep Learning* usa una cascada de múltiples capas de unidades de procesamiento no lineales para la extracción de características.

Las capas “inferiores”, las cuales están más cercanas a los datos de entrada, aprenden características simples mientras que las capas “superiores” aprenden características más complejas derivadas de las características de las capas inferiores [1].

Existen múltiples arquitecturas distintas de redes neuronales que sirven para unos propósitos u otros, como, por ejemplo:

RNN (*Recurrent Neural Network*) son una clase de redes neuronales cuyas conexiones entre neuronas forman un ciclo directo. Al contrario de las redes neuronales *feedforward*, las *RNN* pueden usar su “memoria” interna para procesar una secuencia de entradas, Lo que la hace popular para procesar información secuencial. La “memoria” significa que la *RNN* hace la misma tarea para cada elemento de una secuencia siendo cada salida dependiente de todas las computaciones previas, lo que sería parecido a “recordar” información de lo que se ha procesado hasta el momento [1]. En la **Figura 2** se observa un ejemplo de *RNN* y de *BRNN* [20].

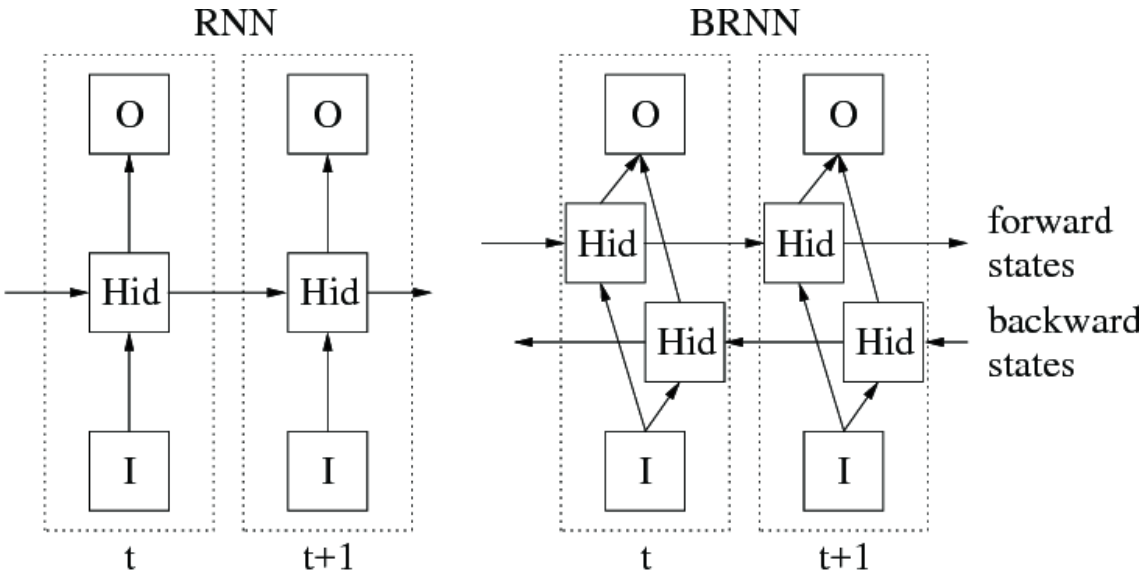


Figura 2: Recurrent RRN y Bidirectional Recurrent RNN.

Otro ejemplo es la red *LSTM*, una red de memoria a corto-largo plazo (del inglés *Long-Short Term Memory*), que es un tipo especial de *RNN* capaz de aprender dependencias a largo plazo. Todas las *RNNs* tienen forma de módulos repetidos en cadena. En las *RNNs* estándar, este módulo que se repite normalmente tiene una estructura simple. Sin embargo, el módulo para un una *LSTM* es más complicado. En lugar de tener una red neuronal de una única capa, hay cuatro capas interactuando de una manera especial. Además, tiene dos estados: estado oculto y estado de célula. En la **Figura 3** se puede observar un ejemplo de red *LSTM* [1].

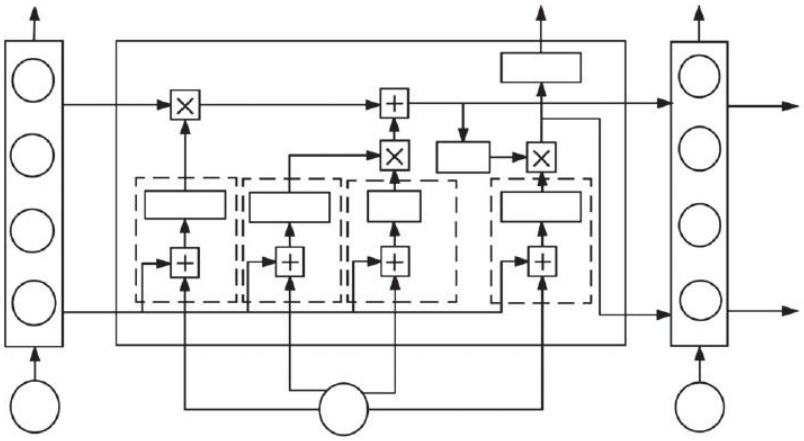


Figura 3: Red LSTM

En el estudio llevado a cabo en [15], se usó una red *Deep belief (Deep Belief Netwok)* junto con un vector de palabras para la detección política en artículos coreanos. El modelo usa una SVM para el cálculo del sesgo, cinco etapas para la detección de sesgos políticos, un *web crawler* desarrollado en Python para recopilar las noticias, KKMA para el análisis de morfemas, el *Word embedding word2vec* y la librería de Python scikit-learn [31]. El *dataset* utilizado contiene 50000 artículos de política desde el 1 de enero de 2014 hasta el 28 de febrero de 2015. Los resultados que obtuvieron fueron: una precisión del 81.8% al predecir correctamente las etiquetas con un error cuadrático medio de 0.120.

3.2.1 Word Embedding

El *Word Embedding* es una técnica para modelar el lenguaje y el aprendizaje de características que transforma las palabras de un vocabulario a vectores numéricos (p.ej. la palabra “perro” -> […, 0.15, 0.541, 0.354, 0.45, …]). La técnica normalmente consiste en, teniendo una representación del vocabulario en un vector de muchas dimensiones y densidad baja (p.ej. un vector *one-hot encoding* que a cada palabra le asigna una dimensión), convertir ese vector en otro de menor dimensión y con mayor densidad. Esta codificación de las palabras hace que cada dimensión del *embedding vector* represente una característica latente de la palabra, por lo que los vectores pueden codificar regularidades lingüísticas y patrones [1].
Este vector, es usado como entrada para el modelo de *Deep Learning* para la extracción del sentimiento que se esté expresando.

Un estudio pionero en el campo del uso de redes neuronales para hacer aprender *Word embeddings* es el hecho por *Bengio et al.* [7]. *Mikolov et al*. Introdujeron el *Continuous Bag-of-Words* (**CBOW**) y el *Continuous Skip-gram* y lanzaron el popular *word2vec* que es usado hoy en día en la mayoría de los trabajos donde se hace uso de *Word Embeddings*.

En [8] y [9] se explica cómo los investigadores crean representaciones semánticas de las palabras mediante el uso de dos técnicas: el *Continuous Bag-of-Words Model* (**CBOW**) y el *Continuous Skip-gram Model* (ambos se muestran en la **Figura 4**). La primera consiste en la representación de cada palabra en función del resto de palabras que la rodean (en el caso de [8], 4 palabras antes de la palabra que se va a representar y cuatro palabras después de la misma). La segunda técnica es parecida a CBOW, pero, en lugar de intentar predecir la palabra actual basándose en el contexto, prueba a maximizar la clasificación de una palabra basándose en otra palabra de la misma frase. Más concretamente, cada palabra se usa como entrada de un clasificador no lineal con una capa de proyección continua, y se intenta predecir las palabras circundantes a la dada como entrada en un cierto rango. Los investigadores encontraron que incrementando el rango aumentaba la calidad de los *Word Embeddings* resultantes, pero también aumenta la complejidad computacional.

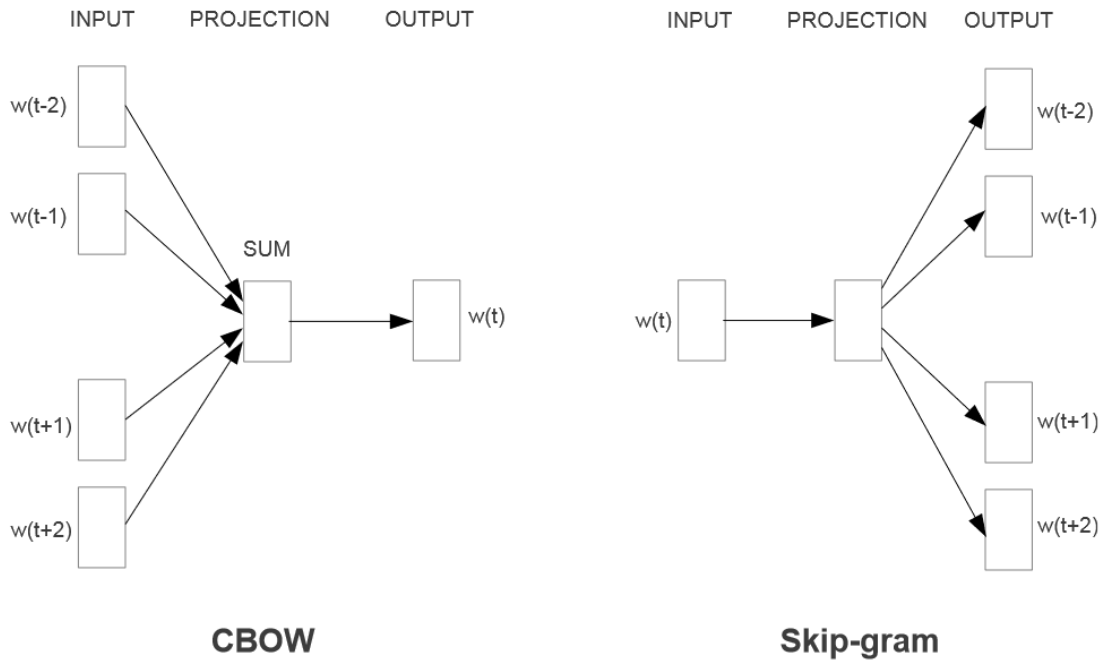


Figura 4: Arquitectura CBOW: predice la palabra actual basándose en el contexto. Arquitectura Skip-gram: predice las palabras circundantes dada la palabra actual [8].

4. Análisis del problema

En esta sección se realizará un análisis del problema a resolver, identificando y analizando las posibles soluciones, y presentando la solución propuesta a dicho problema, justificando las decisiones tomadas.

4.1 Identificación y análisis de soluciones posibles

Para llevar a cabo este trabajo, existen numerosas posibilidades que pasaremos a discutir.

Para determinar el sentido del sentimiento de los discursos primero se debe elegir qué tipo de análisis se debe realizar, en este caso, se optó por el análisis a nivel de documento de la polaridad del sentimiento (es decir, si este es positivo, negativo o neutral) dado que el objetivo es el de analizar gran cantidad de discursos de políticos concretos y ver la evolución de su polaridad en el tiempo. Para determinar el nivel de polaridad de un discurso, se han llevado a cabo numerosas técnicas para la resolución de este problema, algunas de ellas incluyen: el uso de un diccionario que contenga palabras o expresiones asociadas a sentimientos negativos o positivos y, que en base a al número de expresiones negativas/positivas, clasifique. Esta clase de técnicas se desecharon por el uso de otras con mayor uso en el estado del arte actual dentro del ámbito de la clasificación de sentimientos.

Existen también numerosas librerías con clasificadores de sentimiento ya entrenados y listos para usar como es el caso de SpaCy [32]. Mediante esta solución, el usuario simplemente debe introducir un *string* de texto en el modelo de análisis de sentimientos de SpaCy y la librería le devuelve dos valores reales: el nivel de polaridad (comprendido entre -1 y +1 siendo -1 polaridad negativa y +1 polaridad positiva) y el nivel de subjetividad (comprendido entre 0 y 1 siendo 0 un comentario objetivo y 1 un comentario totalmente subjetivo). El problema de esta librería es que está pensada para su uso en ingles por lo que para su implementación hay que traducir cada discurso y esto hace que baje la precisión. Con esta librería, se llevaron a cabo una serie de pruebas antes de trabajar en otras soluciones. Los resultados fueron dispares por lo que se decidió no usarla.

Por todo esto, se decidió el uso de redes neuronales artificiales para lograr modelizar aspectos latentes del análisis de sentimientos en español.

4.2 Solución propuesta

En nuestra solución, usaremos un tipo de red neuronal artificial llamada LSTM para realizar la clasificación de discursos en sentimientos. En primer lugar, se

entrenará la red mediante un conjunto de datos obtenidos de la página del TASS (Taller de Análisis Semántico en la SEPLN) [28] consistentes en *tweets* sobre política española y ya clasificados a nivel de entidad o característica. Una vez entrenada la red neuronal, se procederá a clasificar los discursos mediante ese modelo.
Además, se hará uso de un *Word embedding* entrenado con los propios discursos de los políticos para establecer las relaciones semánticas que existen entre ellos junto con sus partidos.

Para la validación del modelo de *Deep Learning*, se utilizarán la medida de precisión sobre el conjunto de test de *tweets*.

5. Diseño de la solución

Para el diseño de la solución, explicaremos en primer lugar la tecnología utilizada, a continuación, pasaremos a describir la estructura de los datos para su tratamiento y finalmente se muestra la arquitectura y diseño del sistema mediante clases.

5.1 Tecnología utilizada

Como lenguaje de programación se utilizó Python dado que ofrece multitud de librerías de *machine learning*, procesamiento de lenguaje natural y aprendizaje profundo. Además de ser capaz de manipular ficheros y dentro del sistema operativo y tener soporte integrado para la manipulación de ficheros de texto.

Para la descarga de las actas del congreso que se encuentran en formato HTML (o PDF), se utilizó la librería urllib [34] ya incorporada en Python. A continuación, para convertir el objeto html a texto plano se utilizó la librería beautiful soup [35].

Para el uso de expresiones regulares, se utilizó la librería *re* de Python. Para la normalización del texto se utilizó la librería unicodedata.

Para la creación de los *Word Embeddings* y sus operaciones se utilizó la librería Gensim [36] dado que cuenta con una creación y entrenamiento de *Word embeddings* rápida y sencilla; además, cuenta con la implementación del algoritmo FastText [16] [17] usado en este trabajo.

Para la creación y entrenamiento del modelo de aprendizaje profundo, se utilizó la librería Keras que viene instalada como una *API* de TensorFlow [37].

Para la lectura de los *datasets* de *tweets*, se utilizó la librería xml de Python. En concreto xml.dom.minidom [43]. Para su representación, las librerías *Pandas* [42] para su conversión a *dataframes* y matplotlib [41] y Seaborn [44] para su visualización.

5.2 Datos disponibles

Para desarrollar el diseño de la solución, debemos tener en cuenta los datos disponibles para cada tarea en este caso, los *datasets* de *tweets* y las actas del congreso de los diputados.

5.2.1 Tweets

En primer lugar, disponemos de un conjunto de *datasets* proporcionados por la TASS distribuidos en formato XML. Para nuestro problema, utilizaremos los siguientes *datasets*:

- General. Dicho conjunto, se trata de un *dataset* con *tweets* de varias temáticas como política, economía, música, entretenimiento u otros. Nosotros escogimos usar únicamente los *tweets* etiquetados como política o economía españolas para que fuese más ajustado al contexto. El *dataset* está etiquetado con valores de sentimiento "*None*" para indicar que el *tweet* no expresa ningún sentimiento, "NEU" para valor neutral, "P" para positivo, "P+" para muy positivo, "N" para negativo, "N+" para muy negativo. En la Figura 3 se muestra un ejemplo del *dataset* general. Contiene un total de 68000 *tweets* de 150 personalidades conocidas en el mundo hispanohablante sobre cada tema tratado en cada *tweet*. Estos comentarios fueron recogidos en el periodo entre Noviembre de 2011 y Marzo de 2012. [40]

En la **Figura 5** se puede encontrar un ejemplo de uno de los *tweets* que forman el dataset "general-train-tagged".

Figura 5: Muestra de un tweet del dataset "general-train-tagged".

Debido a que nosotros vamos a llevar a cabo una clasificación en tres clases ("P". "NEU" y "N") se han considerado los aspectos muy positivos como si contasen 2 veces un positivo y se ha procedido de la misma manera con el negativo. Para el valor *None* que no indica valor de sentimiento, se clasificó en Neutral. Dicho conjunto consta de los siguientes ficheros:

- general-test1k-tagged formado por 1000 *tweets*.
- general-test-tagged
- general-train-tagged, que, además, también etiqueta si se está de acuerdo con lo que se está diciendo o no mediante la etiqueta

<type> con valores “AGREEMENT” o “DISAGREEMENT”, esta última etiqueta no se tuvo en cuenta para la clasificación del *tweet* en el *dataset* con el resto de los archivos XML.

- politics-test-tagged: También se trata de *tweets* con temática similar al conjunto general, contiene 2500 *tweets* sobre la campaña electoral española para las elecciones generales de 2011. El dataset está etiquetado a nivel de aspecto con valores de sentimiento (<value>) “None” para indicar que el *tweet* no expresa ningún sentimiento, “NEU” para valor neutral, “P” para positivo, “N” para negativo. Además, el *dataset* también etiqueta si se está de acuerdo con lo que se está diciendo o no mediante la etiqueta <type> con valores “AGREEMENT” o “DISAGREEMENT”, esta última etiqueta no se tuvo en cuenta para la clasificación del *tweet* en el *dataset* con el resto de los archivos XML.

- Stompol, la estructura de los archivos “stompol” era bastante distinta a la del resto de *datasets* por lo que se requirió de dos métodos distintos para gestionar cada uno por separado.

  Dicha estructura contiene varios niveles de anidamiento mientras que el resto no. Además, el resto de *datasets* no son únicamente de política sino también de otros temas que hemos querido obviar e incluyen distintas métricas para la medición de la polaridad como el “<type>”, mientras que en los “stompol” se tienen tres niveles de polaridad, el resto de *dataset* clasifican en 5 niveles (menos “politics” que también clasifica a 3).

  En la **Figura 6** se muestra un ejemplo de un tweet, mientras que en la **Figura 7** se muestran las 9 primeras líneas de archivo XML, cada línea de este último fichero corresponde a un *tweet*.

```
<tweet id="591487574071345152">
Mapa con el "batiburrillo" d partidos que ha absorbido
<sentiment aspect="Propio_partido" entity="Ciudadanos" polarity="N">@Ciuda
danosCs
</sentiment>
frente a
<sentiment aspect="Propio_partido" entity="Union_Progreso_y_Democracia" po
larity="P">@UPyD
</sentiment>
que se mantiene #Libres @mpalcedo http://t.co/sYQYv0zSjz
</tweet>
```

Figura 6: Muestra de un tweet del dataset “stompol”.

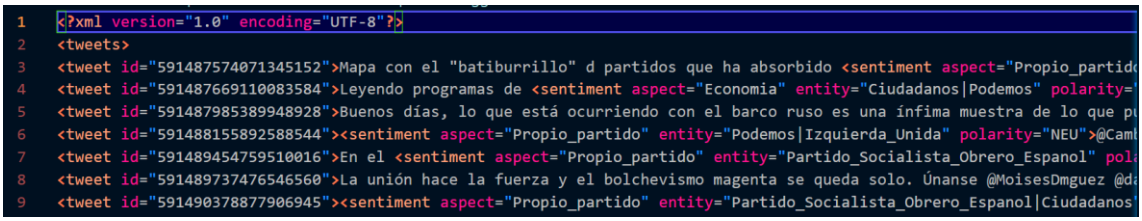


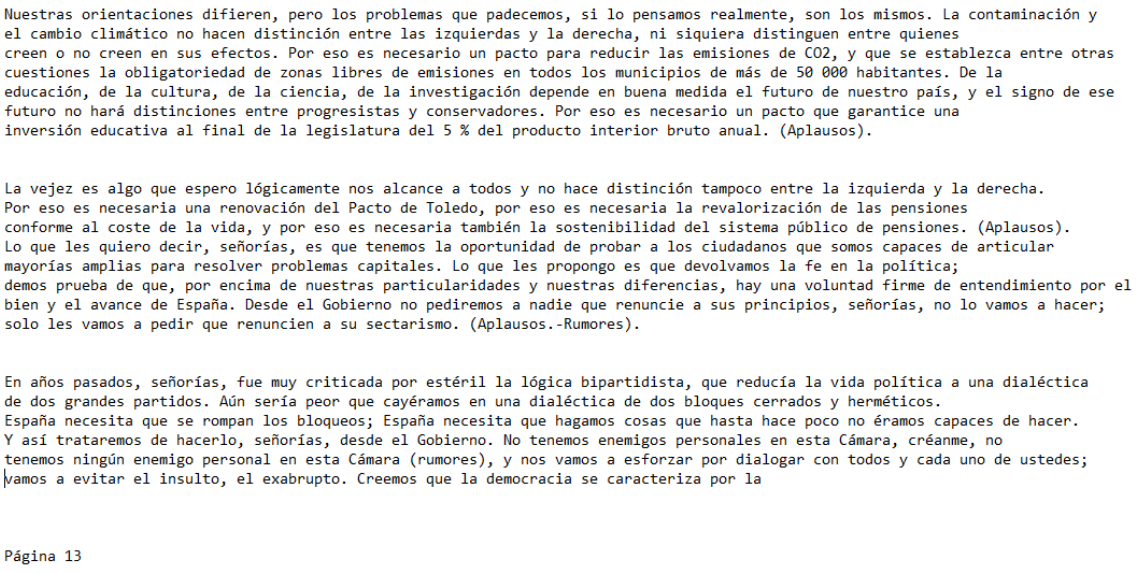
Figura 7: Primeras líneas del dataset "stompol".

Dicho conjunto consta de los siguientes ficheros:

- stompol-test-tagged: Se trata de un dataset con unos 784 *tweets* únicamente sobre política de España. Están clasificados en tres clases "P", "NEU" y "N" a nivel de aspecto [28].
- stompol-train-tagged: Se trata de un dataset con unos 580 *tweets* únicamente sobre política de España. Están clasificados en tres clases "P", "NEU" y "N" a nivel de aspecto [28].

5.2.2 Actas

En el caso de las actas del congreso, se tratan de ficheros HTML alojados en la página web del congreso de los diputados. De este HTML, se ha extraído únicamente la parte de texto correspondiente a las intervenciones de los diputados mediante la librería beautiful soup y se han guardado en memoria como archivos de texto plano. En la **Figura 8** se observa un fragmento de texto de dicha acta.


Nuestras orientaciones difieren, pero los problemas que padecemos, si lo pensamos realmente, son los mismos. La contaminación y el cambio climático no hacen distinción entre las izquierdas y la derecha, ni siquiera distinguen entre quienes creen o no creen en sus efectos. Por eso es necesario un pacto para reducir las emisiones de CO2, y que se establezca entre otras cuestiones la obligatoriedad de zonas libres de emisiones en todos los municipios de más de 50 000 habitantes. De la educación, de la cultura, de la ciencia, de la investigación depende en buena medida el futuro de nuestro país, y el signo de ese futuro no hará distinciones entre progresistas y conservadores. Por eso es necesario un pacto que garantice una inversión educativa al final de la legislatura del 5 % del producto interior bruto anual. (Aplausos).

La vejez es algo que espero lógicamente nos alcance a todos y no hace distinción tampoco entre la izquierda y la derecha. Por eso es necesaria una renovación del Pacto de Toledo, por eso es necesaria la revalorización de las pensiones conforme al coste de la vida, y por eso es necesaria también la sostenibilidad del sistema público de pensiones. (Aplausos). Lo que les quiero decir, señorías, es que tenemos la oportunidad de probar a los ciudadanos que somos capaces de articular mayorías amplias para resolver problemas capitales. Lo que les propongo es que devolvamos la fe en la política; demos prueba de que, por encima de nuestras particularidades y nuestras diferencias, hay una voluntad firme de entendimiento por el bien y el avance de España. Desde el Gobierno no pediremos a nadie que renuncie a sus principios, señorías, no lo vamos a hacer; solo les vamos a pedir que renuncien a su sectarismo. (Aplausos.-Rumores).

En años pasados, señorías, fue muy criticada por estéril la lógica bipartidista, que reducía la vida política a una dialéctica de dos grandes partidos. Aún sería peor que cayéramos en una dialéctica de dos bloques cerrados y herméticos. España necesita que se rompan los bloqueos; España necesita que hagamos cosas que hasta hace poco no éramos capaces de hacer. Y así trataremos de hacerlo, señorías, desde el Gobierno. No tenemos enemigos personales en esta Cámara, créanme, no tenemos ningún enemigo personal en esta Cámara (rumores), y nos vamos a esforzar por dialogar con todos y cada uno de ustedes; vamos a evitar el insulto, el exabrupto. Creemos que la democracia se caracteriza por la

Página 13

Figura 8: Fragmento del acta.

En nuestro caso para el análisis de sentimientos, se utilizaron 42 actas de las sesiones plenarias del congreso comprendidas entre las fechas del 4 de enero de 2020 hasta el 26 de mayo del 2021.

5.3 Arquitectura y diseño detallado del sistema

En esta sección se presenta la arquitectura y diseño del sistema, donde se explicarán las clases utilizadas, la relación entre las mismas, así como las funciones de cada clase en detalle. En la **Figura 9** se puede observar el diagrama de clases UML del sistema.

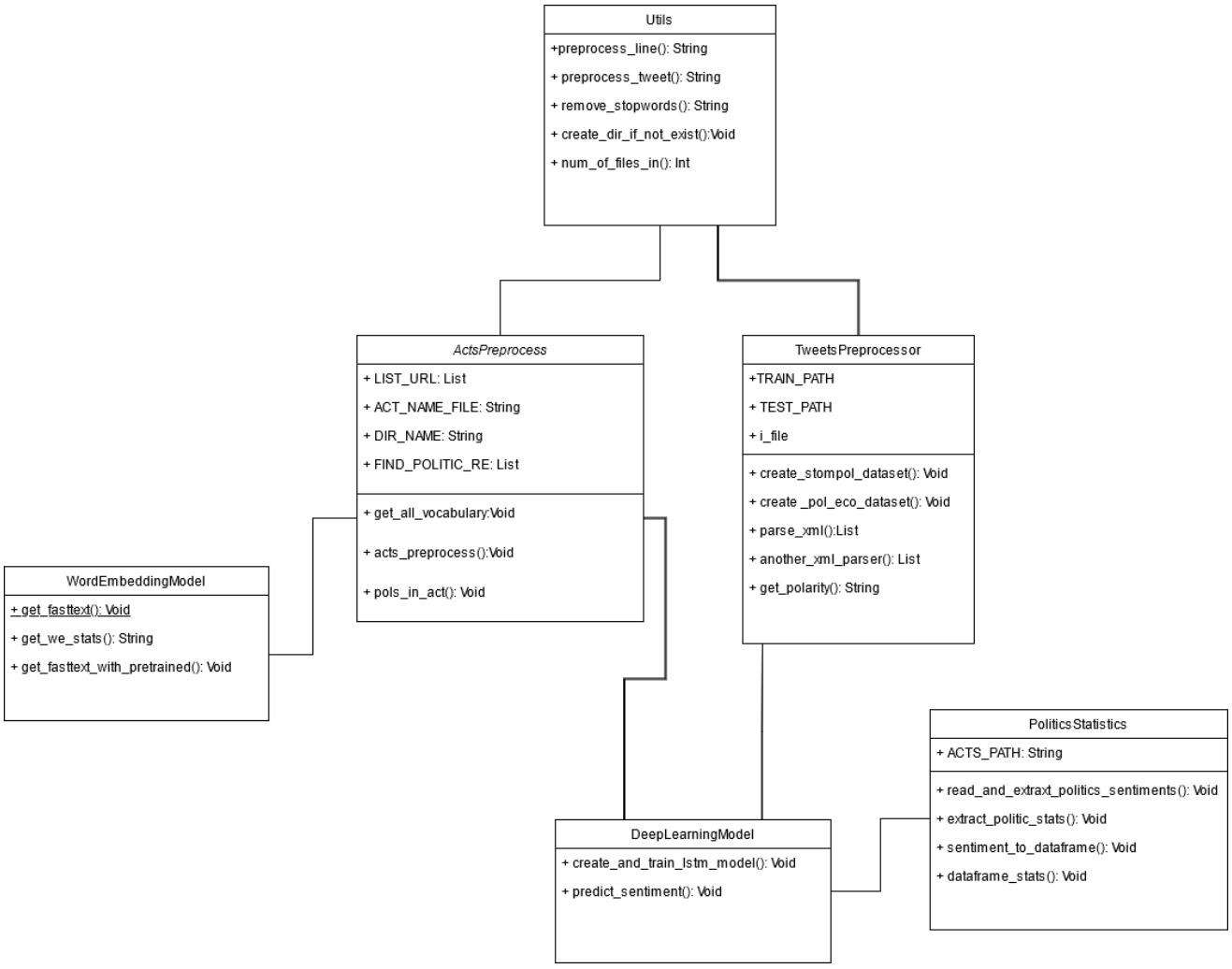


Figura 9: Diagrama de clases.

A continuación, se pasa a explicar la arquitectura y el diseño del sistema.

5.3.1 Clase ActsPreprocessor

En primer lugar, esta es la clase que se encarga del preprocesamiento del texto que se usará posteriormente como entrada para el modelo de aprendizaje profundo.

- La función `acts_preprocess`, abre el archivo de texto plano con las *URLs* correspondientes a cada acta y, por cada *URL* (es decir, por cada acta), crea una carpeta donde guarda un archivo de texto plano *actaCompleta.txt* con el contenido íntegro del acta y llama a la función `pols_in_act`.
- La función `pols_in_act`, recorre el acta descargada en cada carpeta para separarla en archivos de texto plano que contengan los discursos de cada político que interviene en el acta. Para ello, por cada línea del acta la normaliza eliminando mayúsculas, acentos etc. Y se llama a la función `find_politic`, que localiza al político que aparezca en la línea en caso de que lo haga usando una expresión regular. A continuación, se llama a la clase `predict_sentiment(paragraph)` con cada intervención del diputado en el acta que devuelve la polaridad del discurso. Este método pertenece a la clase *DeepLearningModel* que se explicará a continuación. Estos archivos con los discursos de cada diputado se usarán para entrenar el modelo de *Word Embedding* de FastText.
- Por último, el método `get_all_vocabulary`, crea un archivo *AllVocabulary.txt* conteniendo todos los discursos extraídos de las actas y eliminando las *stopwords*. Este archivo será el usado para entrenar el modelo FastText.

5.3.2 Clase TweetsPreprocessor

En segundo lugar, se realiza el preprocesado de los *tweets* que servirán para entrenar el modelo LSTM de clasificación de sentimiento, este proceso se realiza en las funciones:

- La función `create_stompol_dataset`, clasifica los *tweets* en el directorio correcto para que posteriormente puedan ser usados por la librería. Para ello, carga en memoria el dataset elegido (en este caso, los “stompol”) en formato XML y se lee el formato mediante la función `parse_xml`. Dado que no todos los datasets de *tweets* de los que dispone el TASS están construidos mediante la misma estructura, se tuvieron que crear también las funciones `create_pol_eco_dataset` y `another_xml_parser` con la misma función que los anteriores, pero para el resto de datasets. A continuación, se llama a la función `get_polarity` para obtener la polaridad del *tweet* en base a sus etiquetas a nivel de entidad. Finalmente se llama a la función

`preprocess_tweet` de la clase *Utils* que recibe como entrada el *tweet* y lo devuelve habiendo eliminado todo elemento innecesario como emojis, *URLs*, las arrobas de usuarios de Twitter y diversos signos y símbolos.

- La función `parse_xml`, se encarga de abrir el fichero XML correspondiente a un dataset mediante la librería xml.dom.minidom. En esta función se extraen tanto el texto del *tweet* mediante la función `get_tweet_text` como las polaridades que contiene mediante la función `get_tweet_sentiment`. El funcionamiento y la finalidad de la función `another_xml_parser` es similar.
- La función `get_polarity`, clasifica el *tweet* en las clases "P", "NEU" o "N" usando para ello las polaridades obtenidas en la función `get_tweet_sentiment`.

5.3.3 Clase WordEmbeddingModel

La clase *WordEMbeddingModel* se encarga de crear el vector de palabras mediante el algoritmo FastText de la librería Gensim. Esto es llevado a cabo por el método `get_fasttext` que utiliza una clase *MyIter* para obtener el vocabulario del archivo corpus *AllVocabulary.txt*. Una vez creado y entrenado el modelo, lo guarda en memoria para poder ser usado.

A continuación, la función `get_WE_stats` carga el modelo en memora y realiza una serie de cálculos de similitudes semánticas.

Dentro de esta clase, también existe un método `get_fasttext_with_pretrained` que teniendo un vector FastText ya entrenado, lo carga en memoria y lo reentrena para las palabras de *AllVocabulary.txt* que no estén ya representadas en el vector original.

5.3.4 Clase DeepLearningModel

La clase *DeepLearningModel* es la encargada de la creación y entrenamiento del modelo de *Deep Learning* formado por RNN LSTM, además procesa los discursos para introducirlos como entrada al modelo y recoger los resultados.

- La función `create_and_train_LSTM_model`, crea un modelo de redes neuronales con 1 capa oculta formada por LSTM Bidirecional y con 3 neuronas de salida (una por cada clase a clasificar). A continuación, se carga la estructura del *dataset* para crearlo mediante la función `text_dataset_from_directory`
- La función `predict_sentiment`, recibe como entrada el texto de un discurso, carga el modelo de *Deep Learnig* en memoria y tokeniza la entrada mediante `fit_on_text` y `text_to_sequence` ambos de la clase keras.tokenizer. La función devuelve la polarización mediante el uso de `predict`.

5.3.5 Clase PoliticsStatistics

La clase *PoliticsStatistics*, recoge los sentimientos de cada político por acta y obtiene los valores estadísticos de los sentimientos por político para ello, hace uso de las librerías pandas, seaborn y matplotlib. Consta de las siguientes funciones:

- La función `read_and_extract_politics_sentiments`, que extrae los sentimientos de cada acta de cada diputado.
- La función `extract_politic_stats`, que lee de cada discurso la polarización y la inserta en una lista por cada diputado.
- La función `sentiment_to_dataframe` devuelve el *dataframe* correspondiente al discurso de un político con su polarización.
- La función `plot_data` que dibuja la gráfica correspondiente a la polarización de los políticos en un gráfico de barras.
- La función `plot_lines` que dibuja la gráfica de líneas con la polarización por fecha del acta de cada político.

5.3.6 Clase Utils

La clase *Utils* reúne distintas funciones que se han usado a lo largo del trabajo.

Sirve como librería del proyecto, para funciones generales que sirven para pequeñas tareas de procesamiento a nivel de texto, gestión de archivos y directorios, o en general cualquier función que se reutilice en varias clases, sin pertenecer a ninguna en concreto.

- La función `preprocess_line`, devuelve una entrada de texto normalizada.
- La función `preprocess_tweet`, tiene como entrada un *tweet* y lo devuelve sin *URLs*, emojis, arrobas de usuario de Twitter y demás signos de puntuación además de llamar a `preprocess_line`.
- La función `remove_stopwords`, elimina las *stopwords* de la cadena de texto de entrada.
- La función `create_dir_if_not_exist`, crea el directorio de entrada si no existe y si existe lo elimina.
- La función `num_of_files_in`, devuelve el número de ficheros en un directorio.

6. Desarrollo de la solución propuesta

En esta sección pasamos a explicar con más detalle la solución implementada para llevar a cabo el análisis de sentimientos.

6.1 Obtención y preprocesado de los datos

Como se ha comentado anteriormente, las actas se obtienen de la web del congreso de los diputados, siendo descargadas y procesadas por la clase *ActsPreprocess*. Mientras que los *tweets* se obtuvieron del repositorio de *datasets* que mantiene el TASS en su web de cada prueba que realizan, y fueron procesados por la clase *TweetsPreprocess*. En la **Figura 10** se puede observar el flujo de datos seguido por el sistema para la obtención de ambas fuentes de datos. En las secciones posteriores se entrará más en detalle sobre estas cuestiones.

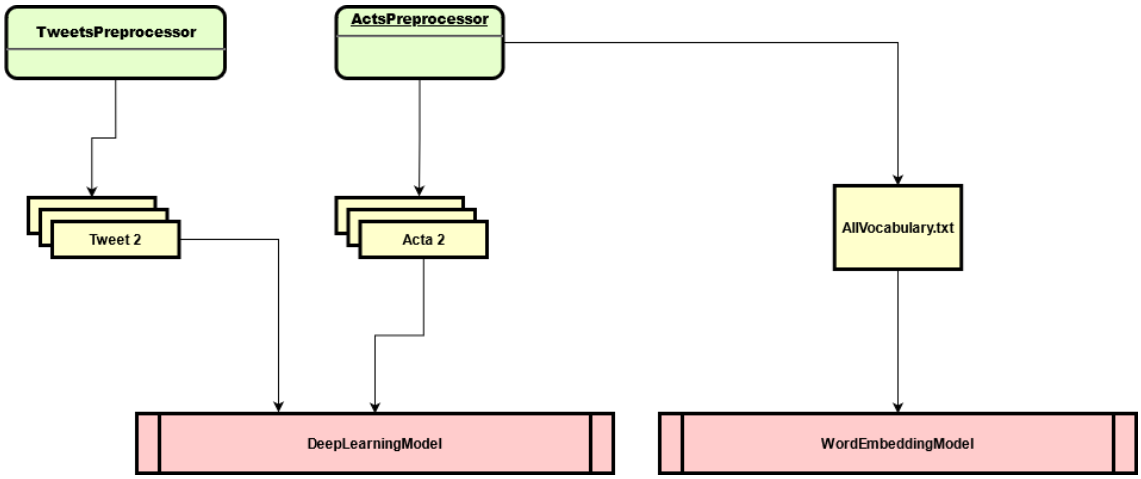


Figura 10: Flujo de datos entre clases.

6.1.1 Preprocesado de las actas

Para desarrollar nuestra solución, vamos a necesitar descargar las actas de las sesiones plenarias del congreso de los diputados las cuales son accesibles desde: https://www.congreso.es/web/guest/busqueda-de-publicaciones.

Para la descarga, la clase *ActsPreprocess* hace uso de la librería beautiful soup para descargar los ficheros en formato HTML de las actas, una vez descargada

el acta, se guarda en el disco en formato .txt (texto plano).
Una vez hecho esto, se pasa a fragmentar el acta completa en las intervenciones de los distintos parlamentarios que intervengan, para ello se debió de buscar un patrón para su automatización. Se encontró que en el momento en que hablaba un diputado, previamente aparecían las expresiones "El señor" o "La señora", a esto le seguía el cargo que el diputado tuviera ya fuera en el gobierno o en la propia cámara.

Sabiendo esto, se recorre el acta descargada línea a línea, se normaliza convirtiendo las mayúsculas en minúsculas y eliminando las tildes, y se comprueba si esa línea contiene los tokens "El señor…"/"La señora…" mediante la siguiente expresión regular:

r"(El señor [A-Z\u00C0-\u017F]+\b)|(La señora [A-Z\u00C0-\u017F]+\b)".

En caso de que se cumpla esa condición, se busca al político dentro de esa línea mediante la función `find_politic(line)`, se elimina el *token* ":" y se limita el tamaño del nombre del archivo a 250 caracteres por poder causar problemas con el sistema de ficheros de Windows. A continuación, se añade al político a la lista de políticos dentro de esa acta y se abre (o crea en caso de no existir) el archivo correspondiente a ese político se escribe en el archivo esa línea que pertenece al discurso del político (esto se sabe mediante una variable de control inicializada a *False* que controla la primera aparición de la intervención de algún político en el acta. Previo a los discursos, el acta contiene otros datos como el tema a tratar o las preguntas de control al gobierno, por ejemplo, en caso de que el acta se tratase de una sesión de control al gobierno) al final del archivo en caso de ya existir.

Una vez se ha encontrado al político en el acta, se sabe que su discurso no se va a ver interrumpido salvo que aparezca otra cadena de texto que cumpla con la expresión regular anterior por lo que cada línea del acta correspondiente al discurso de un mismo político se va acumulando en una variable *paragraph*, que acumula la intervención del político. Una vez la intervención termina, se manda el discurso a analizar mediante la función `predict_sentiment`, de la clase *DeepLearningModel*. La función devuelve la polaridad del discurso y se escribe en el documento de texto la intervención del político y su polaridad.

6.1.2 Preprocesado de los tweets

Para el preprocesado de los *tweets*, se utilizó la librería xml de *Python* para poder cargar en memoria el contenido de los *datasets* de *tweets* clasificados en formato XML. En la **Figura 11** se muestra un diagrama del flujo entre los distintos archivos para la formación del *dataset*.

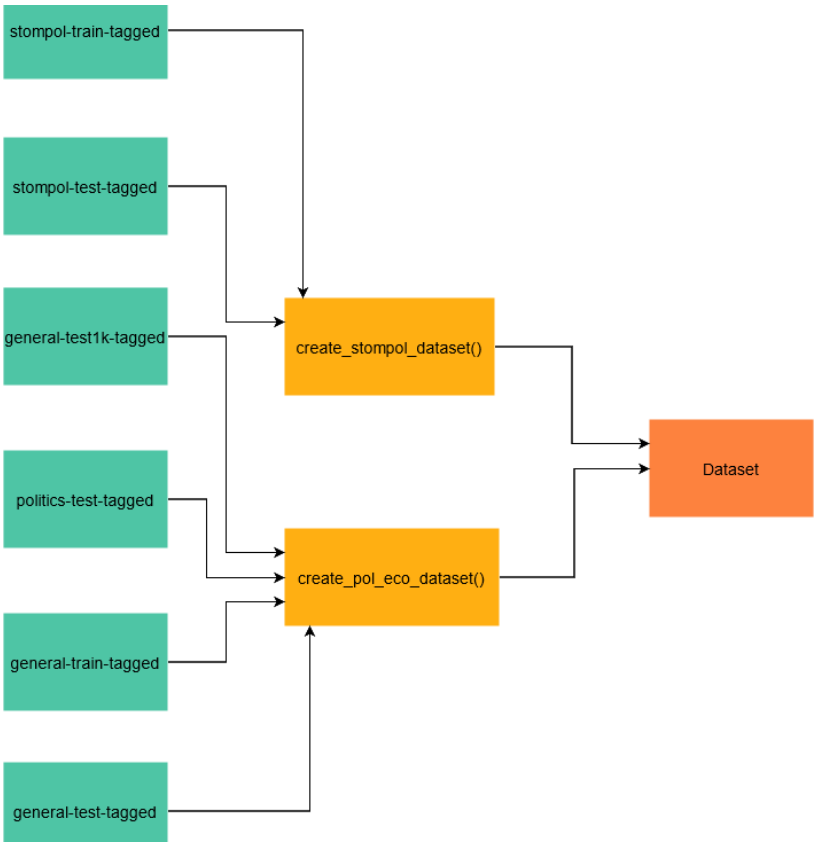


Figura 11: Flujo de datos de los datasets.

En primer lugar, para el procesado de los ficheros "strompol", se llama a la función `create_tweet_dataset`, que es la encargada de llamar a la función `parse_xml` para cargar el *XML* en memoria. En esta función, se obtienen tanto el texto del tweet como las anotaciones sobre los aspectos y sentimientos que están clasificados dentro del *tweet*, la obtención del texto es llevada a cabo por la función `get_tweet_text`, que recorre los nodos del árbol *XML* cargado en memoria y devuelve el texto de estos nodos. A continuación, la función `parse_xml` llama a la función `get_tweet_sentiments` para obtener la clasificación de sentimientos por aspectos. En esta función, por cada *tweet* del árbol *XML*, se busca la etiqueta *sentiment* que es la que contiene las etiquetas del valor de sentimiento. Dentro de esta etiqueta, aparecen las tres etiquetas que definen el sentimiento: *aspect* (que puede ser Economía, Sanidad, Educación, Propio_partido u Otros_aspectos), *entity* (Partido_popular, Partido_Socialista_Obrero_Español, Izquierda_Unida, Podemos, Ciudadanos y Unión_Progreso_y_Democracia) y *polarity* ("P", "NEU" o "N").

Como, por ejemplo, se puede observar en la siguiente sección <sentiment> de un *tweet*:

```
<sentiment aspect="Propio_partido" entity="Ciudadanos" polarity="N">
@CiudadanosCs</sentiment>
```

Para el preprocesado de los *tweets*, se escogieron aquellos que tratasen sobre política y o economía. Para ello, se analizó la etiqueta <topic> para identificar el contexto del *tweet*, aquí tenemos un ejemplo donde el *tweet* hace referencia a temas económicos y políticos:

```
<topics>
   <topic>política</topic>
   <topic>economía</topic>
  </topics>
```

Después de esto, la función `parse_xml` llama a `preprocess_tweet` para eliminar *links*, arrobas, distintos signos de puntuación y *stopwords*.

Finalmente, la función `create_tweet_dataset`, crea los directorios necesarios según se clasifiquen los *tweets* positivos o negativos mediante la función `get_polarity` para que el modelo de *Deep Learning* creado por Keras pueda cargar el directorio y convertirlo en un *dataset* para la tarea de entrenamiento y testeo.

En el caso del procesado del resto de ficheros para la creación del *dataset* conjunto, el proceso es análogo salvo que en lugar de utilizar un XML para entrenamiento y otro para test, se van repartiendo los *tweets* individualmente de cada fichero de manera aleatoria entre los conjuntos de entrenamiento y test, esto es para obtener dos conjuntos relativamente homogéneos en cuanto a número de *tweets* y en cuanto a comentarios.

Para la clasificación en las 3 clases disponibles de cada *tweet*, se realizó mediante la llamada a la función `get_polarity`. En esta función por cada *tweet*, se cuentan el número de aspectos positivo, neutrales y negativos que se encuentran en el *tweet* y se clasifica directamente en una de las clases en caso de existir una clase mayoritaria (esto es, que supere en número al resto).

En caso de no existir una clase mayoritaria (es decir, que haya un empate a 2 o a 3), se calcula la diferencia en valor absoluto entre el número de positivos y negativos y se comprueba si esta diferencia es mayor o igual al número de valores neutrales, en caso de serlo, se clasifica el *tweet* como neutral (el valor predeterminado) si “P” = “N” > “NEU” o si “P” = “N” = “NEU”. En caso de que haya un empate a 2 entre “P” y “NEU” o entre “N” y “NEU”, se clasifica en el valor de polaridad que no sea neutral. En la **Figura 12** se muestra el algoritmo.

```python
def get_polarity(self, polarity):  # (aspect, identity, polarity):
    """
    3 primeros ifs ->
        si hay un claro ganador se asigna a ese sino -> ultimo if:
            si hay empate a dos (o a 3):
                P = 0; N = 1; NEU = 1 -> se le obliga a elegir entre positivo o negativo
                P = 1; N = 1; NEU = 0 -> NEU
                P = 1; N = 1; NEU = 1 -> NEU # Valor pol = "NEU" predeterminado
    """
    pos = len([p for p in polarity if p == "P"])
    pos_plus = len([p for p in polarity if p == "P+"])
    neg = len([n for n in polarity if n == "N"])
    neg_plus = len([n for n in polarity if n == "N+"])
    neu = len([n for n in polarity if n == "NEU"])
    pol = "NEU"
    pos += 2 * pos_plus
    neg += 2 * neg_plus
    pos_neg = pos - neg
    # max_pnn = max(pos, neg, neu)
    if pos > neg and pos > neu:
        pol = "P"
    elif neg > pos and neg > neu:
        pol = "N"
    elif neu > neg and neu > pos:
        pol = "NEU"
    elif abs(pos_neg) >= neu:
        pol = (
            "NEU"
            if pos == neg
            else "P"
            if pos_neg > 0
            else "N"
            if pos_neg < 0
            else "NEU"
        )
    return pol
```

Figura 12: Algoritmo de cálculo de polaridad por aspecto.

Para la creación de nuestro *dataset* total, se dividieron los archivos obtenidos del TASS aleatoriamente en dos partes del mismo tamaño aproximadamente para el conjunto de entrenamiento y para el conjunto de test resultando finalmente en un *dataset* formado por 2871 *tweets* negativos, 2427 neutrales y 1760 positivos para el conjunto de entrenamiento (7058 en total) y 2785 *tweets* negativos, 2137 neutrales y 1784 negativos (6706 en total) formando un *dataset* de 13764 *tweets*.

6.2 Word Embedding FastText

En la primera parte del análisis del discurso queremos conocer qué conceptos están más semánticamente relacionados con cada uno de los diputados que hemos elegido, además, queremos saber que palabras están relacionadas con los nombres de cada gran partido con representación parlamentaria. Calcularemos la distancia entre cada diputado para establecer su distancia mediante la similitud coseno. A continuación, para cada concepto, se calcularán

las palabras más cercanas semánticamente primero con el vocabulario entero (con las actas de todos los diputados).

Para ello, utilizaremos una herramienta conocida como *Word Embedding* explicada anteriormente. En este caso se ha escogido la librería Gensim que tiene una implementación propia del modelo FastText [16]. Una de las características clave de la representación de palabras de FastText es su capacidad para producir vectores para cualquier palabra, incluso las inventadas. De hecho, los vectores de palabras de FastText se construyen a partir de vectores de subcadenas de caracteres que contiene. Esto permite construir vectores incluso para palabras mal escritas o concatenación de palabras. Este modelo es particularmente bueno con las derivaciones de las palabras y ha demostrado tener buenos resultados en diferentes idiomas al inglés [38].

El modelo se ha implementado en Python con la librería Gensim, en primer lugar, se construye el modelo con un tamaño de vector de 300 dimensiones, este número indica el número de características que el vector está representando por cada palabra. Se escoge una ventana de 5 palabras para que en la representación de una palabra se tengan en cuenta las 5 palabras que la rodean y se establece que una palabra tenga representación si aparece como mínimo 1 vez en el vocabulario.

En segundo lugar, se construye el vocabulario mediante una clase iterable que extrae cada palabra del corpus formado por todas las actas, y la introduce al vocabulario del modelo. Con el modelo y el vocabulario ya construidos, se procede a entrenar el modelo con el vocabulario y estableciendo que el entrenamiento itere 10 veces sobre el mismo vocabulario para que el modelo establezca las relaciones semánticas.

Una vez ya entrenado el modelo, se guarda en disco, guardando en disco 3 archivos, “fasttext.model.syn1neg.npy”, “fasttext.model.wv.vectors_ngrams.npy” ambos de 53MB y “fasttext.model.wv.vectors_vocab.npy” de 2.3GB.

Una vez creado el Word embedding, este modelo se carga en memoria y se calculan las siguientes similitudes semánticas:

Primero se calculan las similitudes que tienen los nombres de cada líder político de cada uno de los 5 grandes partidos en este caso, Pedro Sánchez, Pablo Casado, Santiago Abascal, Pablo Iglesias e Inés Arrimadas.
Además de ello, también se calculan las palabras más similares a las siglas y nombres de los partidos políticos, además de la distancia coseno que existe entre cada uno de ellos.

6.3 Modelo LSTM

Para la realización del modelo de aprendizaje profundo, utilizaremos la librería Keras. En la **Figura 2** se muestra una imagen de un *Bidirectional RNN*, a la izquierda, y un modelo profundo bidireccional RNN a la derecha [1].

En primer lugar, para crear el modelo, creamos una capa de vectorización de texto que va a convertir nuestro *dataset* de texto en un *dataset* formado por los 80000 *tokens* de texto más frecuentes en el *dataset* en formato de enteros. A continuación, se introducen los *tweets* codificados en *integers* en una capa de *Embedding* que va a devolver un vector de *embedding* de dimensión 300 por cada palabra que va a ser introducida en el modelo.

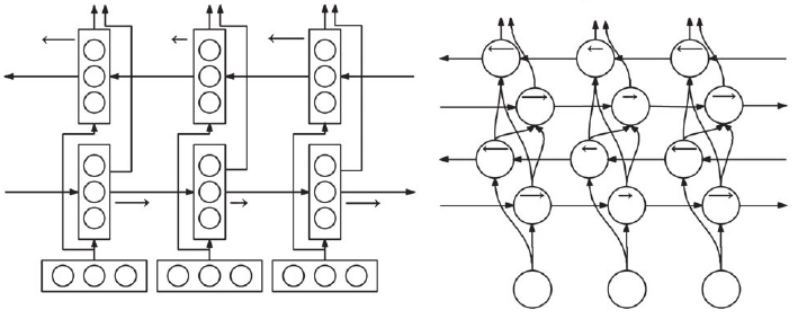


Figura 13: Modelo bidireccional RNN, a la izquierda, y modelo profundo bidireccional RNN a la derecha.

La primera capa LSTM Bidireccional (**Figura 13**) formada por 128 neuronas, recibe el vector de *embedding* y devuelve el valor de la entrada procesada a través de sus pesos a la siguiente capa. Esta última capa es de tipo dense (esto es, que todas las salidas de la capa anterior están conectadas a todas las entradas de esta capa) formada por 3 neuronas activadas mediante la función "*softmax*" (se escogió esta función de activación ya que estamos ante un problema de clasificación multi etiqueta) que nos van a devolver la salida de nuestro modelo: un tensor con una probabilidad de pertenencia a cada clase por cada discurso de entrada que le pasemos, de la forma:

$$[P_{doc\text{-}1}(\text{“P”}), P_{doc\text{-}1}(\text{“NEU”}), P_{doc\text{-}1}(\text{“N”})]$$

A continuación, para entrenar el modelo, cargamos tanto el *dataset* de entrenamiento como el de *test* en memoria mediante la función `text_dataset_from_directory`, para así convertir la estructura de ficheros que hemos generado con la clase *TweetPreprocess* a un objeto de tipo *Dataset*, así el modelo podrá operar con él. Almacenamos en una variable de tipo *Dataset* los textos del conjunto de entrenamiento sin las etiquetas y a continuación llamamos a la función `adapt` sobre el *Dataset* de los textos de entrenamiento. Esta función adapta el *Dataset* a la capa desde la que la hemos llamado, en

este caso estamos usando una capa *vectorize_layer* que adapta el *Dataset* para el número máximo de palabras que se van a tener en cuenta (en este caso, se tendrán en cuenta para la creación del vocabulario las 80000 palabras más frecuentes del *Dataset*), su salida será una secuencia de *integers* que representen cada *token* del *Dataset* y con una dimensionalidad de salida de 300 dimensiones.

Una vez tenemos tanto los datos de entrenamiento como los de test vectorizados, se crea un *buffer* que almacena en memoria cache una cantidad de los datos que van a ser entrenados para mejorar la velocidad del entrenamiento al estar cargados ya en este tipo de memoria ya que van a ser cargados por la tarjeta gráfica cuyos tiempos de carga y de cálculo son costos, se hace lo mismo con el conjunto de test. En la **Figura 14** se puede observar el grafo principal del modelo Bidireccional LSTM, generado por TensorBoard [21].

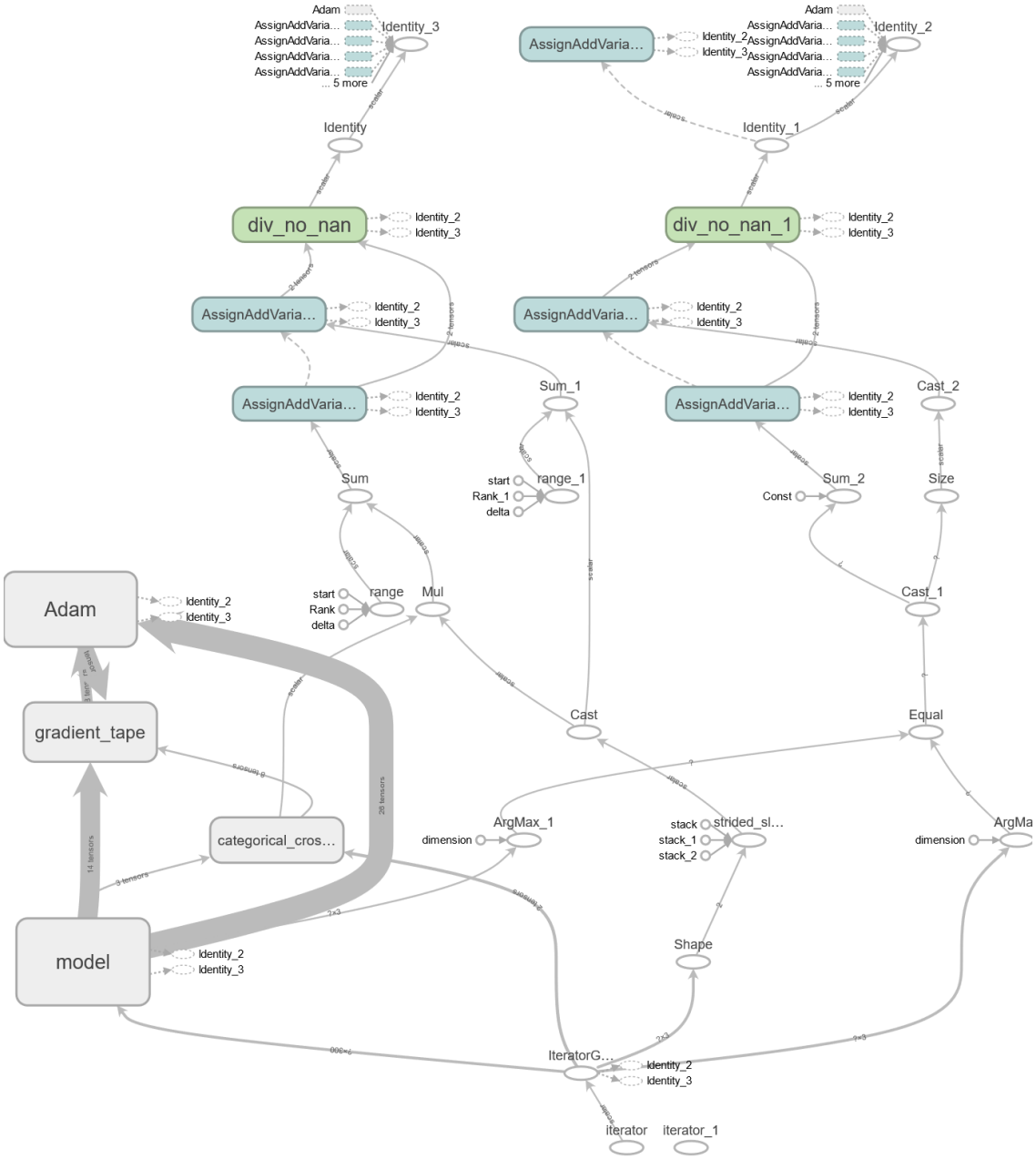


Figura 14: Grafo principal del modelo Bidireccional LSTM.

Seguidamente se crea el modelo con la función de optimización "*Adam*" el cual es un método "*computacionalmente eficiente, tiene pocos requerimientos de memoria, es invariante frente al reescalado diagonal de gradientes y funciona bien para problemas extensos en términos de datos o parámetros*" Kingma et al., 2014 [39].

Una vez creado el modelo para su entrenamiento se llama a la función `fit` sobre el modelo que recibe como parámetros el conjunto de textos de entrenamiento en forma de *Dataset* numérico, el número de iteraciones y los *callbacks* en caso de que queramos llamar a una función durante el paso del

entrenamiento. En este caso en el *callback* se ha llamado a una función que crea el modelo TensorBoard del modelo de redes neuronales para su mejor evaluación.

Finalmente, para evaluar el rendimiento del modelo de *Deep Learning*, se llama a la función `evaluate` con el conjunto de *test* para que nos devuelva los valores de precisión y *loss* del modelo frente al conjunto de test.

En la **Figura 15** se observa la *accuracy* del modelo calculada por cada *epoch*, donde se ve que la precisión aumenta para cada *epoch*. Mientras que en la **Figura 16** se observa el *loss* durante cada *epoch*, que forma una tendencia descendente. Lo que implica que el modelo va mejoran a medida que se realizan más *epoch*, alcanzando el mejor resultado a partir de la iteración 4 con una *accuracy* del 99.53% y una *loss* de 0.0167 sobre el conjunto de entrenamiento. El modelo alcanzó un 92.50% de *accuracy* y un *loss* de 0.3245 sobre el conjunto de test.

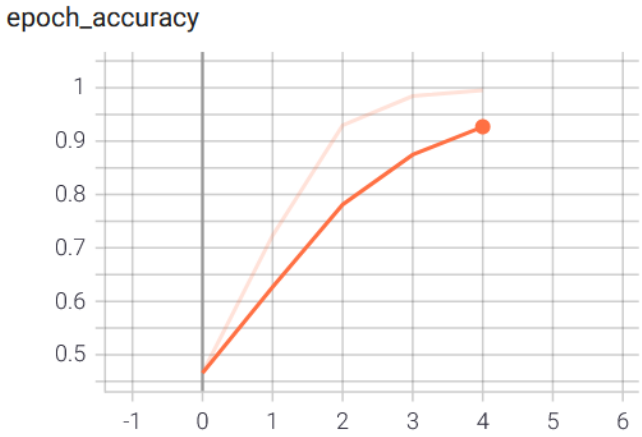


Figura 15: Precisión sobre el conjunto de entrenamiento por cada epoch.

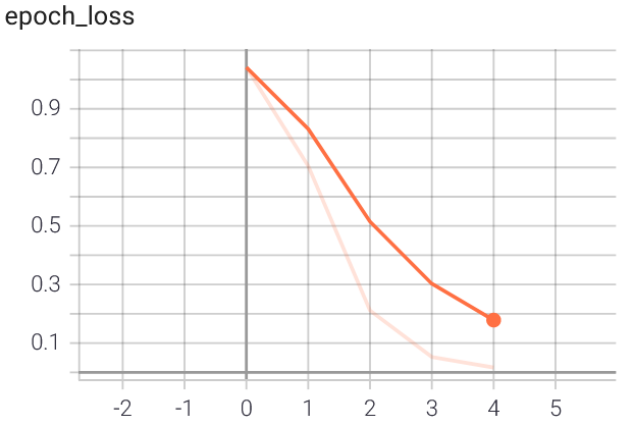


Figura 16: Loss sobre el conjunto de entrenamiento por cada epoch.

7. Experimentación y resultados

En el siguiente apartado, se incluyen las pruebas que se realizaron para la obtención de los resultados.

7.1 Modelo Word Embedding FastText

Los cálculos de similitud entre palabras se muestran como una tupla (palabra, distancia). A continuación, se muestran los resultados obtenidos para los cálculos previamente mostrados.

Tabla 1: Distancias "Pedro Sánchez".

Token	Distancia
'sanchez'	0.949
'manche'	0.729
'candidez'	0.688
'carabanchel'	0.684
'echenique'	0.683
'rajoy'	0.658
'sancho'	0.654
'candidato'	0.652
'ramirez'	0.643
'abascal'	0.642

En el caso de Pedro Sánchez, cuyos resultados se muestran en la **Tabla 1**, es interesante notar que el modelo ha sido capaz de captar que, durante parte de las actas, Pedro Sánchez ha sido el candidato a la presidencia del gobierno en las sesiones de investidura. Además, aparecen los nombres de "Echenique" (El portavoz en el congreso del partido con el que comparte gobierno) y su antecesor en el puesto de presidencia "Rajoy". Además, aparece el nombre de "Abascal" mostrando las numerosas referencias de Pedro Sánchez hacia el diputado. No hay referencia al líder de la oposición Pablo Casado como si ocurría en [22].

Tabla 2: Distancias “Pablo casado”.

Token	Distancia
‘envasado’	0.976
‘pisado’	0.969
‘avisado’	0.968
‘osado’	0.966
‘rehusado’	0.965
‘arrasado’	0.964
‘usado’	0.963
‘engrasado’	0.963
‘tasado’	0.962
‘cursado’	0.960

En el caso de Pablo casado, **Tabla 2**, no se encontraron similitudes semánticas a simple vista.

Tabla 3: Distancias “Santiago Abascal”.

Token	Distancia
‘abascal’	0.903
‘mataraabascal’	0.902
‘bal’	0.862
‘pascal’	0.845
‘mariscal’	0.822
‘berrocal’	0.795
‘santiago’	0.770
‘verba'	0.770
‘cabal’	0.765
‘cal’	0.764

En el caso de Santiago Abascal, **Tabla 3**, llama la atención que el modelo haya obtenido el token “mataraaabascal” esto es debido a un *hashtag* que se hizo viral en Twitter con ese mismo texto y fue comentado en las sesiones del congreso de los diputados. Además de ello, el modelo a representado a “Mariscal” como cercano siendo ambos del mismo partido.

Tabla 4: Distancias "Pablo Iglesias".

Token	Distancia
'iglesias'	0.956
'eclesiastes'	0.773
'simancas'	0.758
'cartesianas'	0.734
'abascales'	0.731
'sayas'	0.725
'fantasias'	0.724
'sanchistas'	0.722
'monjas'	0.718
'monarcas'	0.716

Para Pablo Iglesias, **Tabla 4**, se ha representado a "Simancas" refiriéndose a Rafael Simancas, un político socialista. Además, aparecen las palabras "Sanchistas" debido a la relación entre Pablo Iglesias y Pedro Sánchez. Además, aparece "Sayas" refiriéndose probablemente a Sergio Sayas del partido Unión del Pueblo Navarro. Por último, han llamado la atención la palabra "Eclesiastes" así que se decidió buscar en el corpus usado por el *Word Embedding* y se encontró la siguiente intervención del diputado Garces Sanagustin del Partido Popular donde hace referencia al "Vicepresidente" (sin nombrar directamente a Iglesias):

"*No confunda la moderación con la contundencia. He podido utilizar un tono probablemente menos vehemente que otras veces, pero, fíjese, aquí un* ***vicepresidente*** *dice lo que dice con un tono absolutamente lineal y es una de las mayores agresiones verbales a la democracia que se han escuchado en los últimos cuarenta años, ustedes pactan con EH Bildu en un tono absolutamente neutral y eso si que es radicalidad. (Aplausos). El tono será el que tenga que utilizar. O el tono que escuchamos ayer cuando el* ***eclesiastés*** *del fondo norte nos empezó a insultar a todos los diputados del Partido Popular. Creo que hay que buscar un equilibrio, evidentemente, y buscar la fórmula necesaria para poder tener, desde luego, el equilibrio dialectico necesario. (Aplausos)."*

Líneas después, confirma que se refiere al vicepresidente: *"… cuando ha dicho el* ***vicepresidente*** *que nos tenía que echar del Estado..."*

Tabla 5: Distancias "Inés Arrimadas".

Token	Distancia
'animadas'	0.946
'arrimadas'	0.943
'barriadas'	0.914
'abrumadas'	0.913
'arrastradas'	0.912
'arriesgadas'	0.911
'arrasadas'	0.909
'toneladas'	0.907
'malvadas'	0.905
'cansadas'	0.905

En el caso de Inés Arrimadas, mostrado en la **Tabla 5**, no se ha logrado establecer una relación más allá de la sintáctica. También se analizaron las palabras más cercanas a los partidos políticos como en el caso de [22].

Para "PSOE", en el caso de la nube de palabras de [22] se obtuvieron: "proyecto", "socialista", "derecha" y "futuro". Para nuestro caso mediante el uso de *Word embeddings* y con los *tokens* "PSOE" y "Partido Socialista Obrero Español", en nuestro caso, se muestran en la **Tabla 6**.

Tabla 6: Distancias para "PSOE" y "Partido Socialista Obrero Español".

Token "PSOE"	Distancia	Token "Partido Socialista Obrero Español"	Distancia
'dificil'	0.503	'socialistacomunista'	0.804
'breve'	0.483	'marginalista'	0.795
'clara'	0.476	'socialista'	0.794
'diferira'	0.448	'regionalista'	0.793
'propositiva'	0.437	'moralista'	0.784
'acompleje'	0.436	'foralista'	0.782
'facil'	0.435	'patrimonialista'	0.782
'acomplejada'	0.425	'federalista'	0.781
'complicada'	0.423	'analista'	0.779
'compleja'	0.422	'avalista'	0.779

En primer lugar, cabe destacar que, en el caso de las siglas, al utilizar también la distancia sintáctica para su cálculo, arroja valores bastante bajos de similitud, véase que la similitud máxima que se ha encontrado entre todos los discursos arroja un valor de 0.5.

Como se puede observar, aparecen numerosos conceptos ligados al partido socialista como pueden ser "socialista", "regionalista", "foralista" o "federalista" en el ámbito territorial. También aparecen conceptos que puede haber utilizado

la oposición como puede ser “moralista”. Además de esto, aparece la palabra “acomplejada” usada por miembros del partido socialista en numerosas ocasiones para referirse a la “*derecha acomplejada del PP*” frente a VOX como por ejemplo en el siguiente fragmento de Elorza González, diputado del PSOE: “*Cuando la extrema derecha y la derecha acomplejada acusan…”.* Además de esto, aparecen “socialistacomunista” como calificativo recurrente que usa Vox para el gobierno.

Para “PP”, en el caso de [22], aparecían: “Sánchez”, “PSOE” y “Madrid”, en nuestro caso mostramos la **Tabla 7**.

Tabla 7: Distancias para “PP” y “Partido Popular”.

Token “PP”	Distancia	Token “Partido Popular”	Distancia
'juridicas'	0.471	'impopular'	0.798
'universitarias'	0.460	'manipular'	0.785
'organicas'	0.451	'popular'	0.762
'penitenciarias'	0.437	'reformular'	0.719
'lenguas'	0.436	'postular'	0.715
'autoritarias'	0.432	'angular'	0.709
'ordenanzas'	0.405	'rotular'	0.707
'organizativas'	0.400	'simular'	0.706
'literarias'	0.398	'emular'	0.704
'nanoparticulas'	0.396	'deambular'	0.696

Para el caso del Partido Popular, no se han podido establecer relaciones semánticas obvias, únicamente parecen existir similitudes sintácticas.

Para “VOX” en [22] obtuvieron: “Cataluña”, “Barcelona”, “españoles” y Europa “, en nuestro caso mostramos los resultados en la **Tabla 8**.

Tabla 8: Distancias para “VOX”

Token “VOX”	Distancia
'debera'	0.212
'limbo'	0.204
'aluminio'	0.203
'elimino'	0.202
'reabrio'	0.202
'deberse'	0.201
'bioenio'	0.197
'merino'	0.196
'debiera'	0.195
'imperio'	0.194

En el caso de Vox podemos observar que la palabra con mayor similitud es “deberá” con una similitud muy baja muy probablemente debido a que establecer una relación sintáctica entre la palabra “Vox” y la mayoría de las palabras de uso común en castellano es más complejo que en el resto de los casos.

Tabla 9: Distancias para “Podemos”.

Token “Podemos”	Distancia
'enredemos'	0.954
'blindemos'	0.951
'dudemos'	0.947
'demos'	0.941
'cuidemos'	0.933
'ayudemos'	0.927
'poseemos'	0.926
'defraudemos'	0.925
'gocemos'	0.92
'nacemos'	0.92

Para “Podemos”, según [22]: “derecho” y “gente”, en nuestro caso, mostrado en la **Tabla 9**, la segunda palabra con mayor similitud es “blindemos” usada frecuentemente por el partido para referirse a “blindar las pensiones” u otros servicios públicos que ellos consideren. Además, se incluyen palabras como “cuidemos” y “ayudemos” muy usadas por podemos para referirse a algún sector de la población.

Tabla 10: Distancias para “Ciudadanos”.

Token “Ciudadanos”	Distancia
'ciudadanos'	0.988
'conciudadanos'	0.950
'peldaños'	0.875
'daños'	0.844
'ciudadano'	0.819
'mejicanos'	0.807
'afganos'	0.806
'prusianos'	0.799
'espartanos'	0.799
'copernicanos'	0.796

Para “Ciudadanos”, en el caso de [22] obtuvieron “Sánchez”, “Cataluña” y “PSOE”, en nuestro análisis, **Tabla 10**, el primer y el segundo *token* hacen referencia al propio partido. No se han podido establecer otras relaciones que no sean sintácticas.

Para el token “Izquierda”, se muestran los resultados en la **Tabla 11**, mientras que para “Derecha”, se muestran en la **Tabla 12**.

Tabla 11: Distancias para “Izquierda”.

Token “Izquierda”	Distancia
'izquierda'	0.982
'ultraizquierda'	0.942
'izquierdista'	0.887
'ultraizquierdista'	0.868
'izquierdo'	0.854
'centroizquierda'	0.835
'mierda'	0.788
'pierda'	0.749
'izquierdas'	0.738
'delinquiera'	0.729

Tabla 12: Distancias para “Derecha”.

Token “Derecha”	Distancia
'ultraderecha'	0.937
'derecha'	0.914
'ultraultraderecha'	0.911
'brecha'	0.891
'maltrecha'	0.867
'pecha'	0.84
'ultraderechista'	0.809
'echa'	0.802
'ultraderechas'	0.795
'acecha'	0.787

Para el caso de izquierda y derecha, destaca que, en el caso de la derecha, arroje mayor similitud con “*ultraderecha*” (0.94) que con la propia palabra “*derecha*” (0.91); esto puede deberse a la percepción general que tiene la cámara hacia el concepto de derecha de asociarlo con un concepto más extremista que centrado (muy probablemente debido a la preeminencia en número de representación de izquierdas, véase la figura “Evolución ideológica” abajo). Esto también explicaría la aparición de la palabra “centroizquierda” entre las similitudes para “izquierda” con una similitud del 83.4%, mientras que entre las similitudes de la “derecha” no aparece la palabra “centroderecha” (por lo menos con similitudes mayores al 78.67%). Además, en la derecha, aparece el caso de la “ultraultraderecha”, pronunciado por el presidente del gobierno durante su discurso de investidura:

“…dinamicas reaccionarias, de regresion y de involucion que, por desgracia, estamos viendo en algunas comunidades autonomas y en algunos ayuntamientos como consecuencia de ese entendimiento entre la ultraderecha y la ***ultraultraderecha****, ademas de la derecha.”,* entre otros, mientras que no hay caso similar en la izquierda.

Entre la izquierda, aparecen también palabras como “*mierda*”, “*pierda*” o “*delinquiera*” como en la derecha “*brecha*”, “*maltrecha*” o “*acecha*” que tienen mayor probabilidad de aparecer debido a su similitud sintáctica.

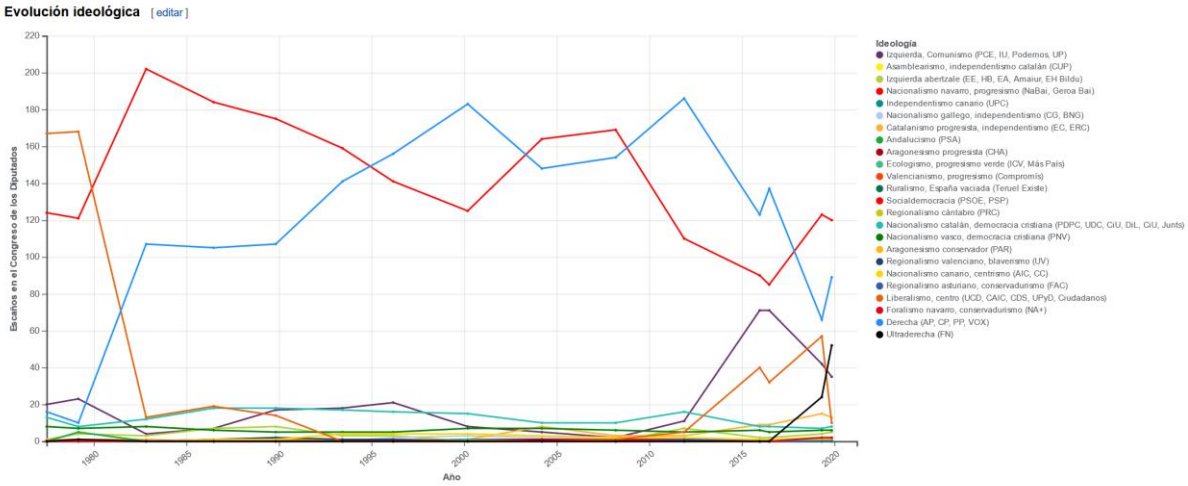


Figura "Evolución ideológica", donde se observa que el conjunto "Socialdemocracia" formado por PSOE y PSP es el que más escaños obtiene seguido de la "Derecha (AP, CP, PP, VOX)". Fuente: https://es.wikipedia.org/wiki/Congreso_de_los_Diputados#Evoluci%C3%B3n_ideol%C3%B3gica.

Tabla 13: Distancias para "Pandemia".

Token "Pandemia"	Distancia
'pandemia'	0.983
'pospandemia'	0.97
'prepandemia'	0.963
'epidemia'	0.941
'academia'	0.921
'pandemica'	0.88
'anemia'	0.861
'temia'	0.843
'pandemias'	0.801
'crisis'	0.797

Para "Pandemia", mostrada en la **Tabla 13**, llamó la atención que una de las palabras que FastText relacionase con pandemia sea "crisis" en referencia a la crisis sanitaria y económica provocada por la misma.

Tabla 14: Distancias para "Covid19".

Token "Covid19"	Distancia
'covid'	0.876
'poscovid'	0.852
'datacovid'	0.845
'coronavirus'	0.806
'poscoronavirus'	0.804
'virus'	0.69
'asistenciacovid'	0.667
'virgili'	0.655
'pandemia'	0.645
'video'	0.632

En el caso de "covid19", mostrado en la **Tabla 14**, el modelo logró relacionarla tanto con la palabra "coronavirus" como con "virus". Además, logró establecer la relación con la palabra "pandemia".

En cuanto al cálculo de la similitud entre palabras, se usó la función *n_similarity* que devuelve la similitud coseno (un valor entre -1 y 1 siendo -1 antónimos y 1 sinónimos) entre dos palabras dadas. Para ello se calcularon una serie de distancias presentadas a continuación.

Tabla 15: Distancias entre las siglas de los partidos.

	PSOE	PP	VOX	PODEMOS	CIUDADANOS
PSOE	-	0.060	-0.081	0.108	0.021
PP	0.060	-	-0.074	-0.098	-0.118
VOX	-0.081	-0.074	-	0.009	0.015
PODEMOS	0.108	-0.098	0.009	-	-0.148
CIUDADANOS	0.021	-0.118	0.015	-0.148	-

En la **Tabla 15**, se muestran las distancias entre las siglas de los cinco grandes partidos calculadas mediante la librería Gensim.

Para el PSOE, la mayor similitud se alcanzó con Podemos mientras que la mínima fue con Vox. Para el caso del PP, su máxima similitud fue con el PSOE mientras que la mínima con Ciudadanos. En cuanto a Vox, alcanzó su máximo con ciudadanos y su mínimo con el PSOE. Podemos tiene su máxima similitud con el PSOE y su mínima con Ciudadanos y, como ya hemos dicho, Ciudadanos alcanza la máxima con Vox y la mínima con Podemos.

En cuanto a los políticos, se calculó su distancia coseno y los resultados obtenidos fueron los mostrados en la **Tabla 16**.

Tabla 16: Distancias entre nombres de políticos.

	Sánchez	**Casado**	**Abascal**	**Iglesias**	**Arrimadas**
Sánchez	-	0.431	0.518	0.523	0.110
Casado	0.437	-	0.395	0.471	0.315
Abascal	0.528	0.395	-	0.480	0.147
Iglesias	0.545	0.471	0.480	-	0.100
Arrimadas	0.103	0.315	0.147	0.100	-

La correlación entre Sánchez e Iglesias se mantiene además de la diferencia entre Ciudadanos y Podemos.

Dentro de esta clase, también existe una función `get_fasttext_with_pretrained` que teniendo un vector FastText ya entrenado, lo carga en memoria y lo reentrena para las palabras de *AllVocabulary.txt* que no estén ya representadas en el vector original. El uso de este método se descartó principalmente por el alto coste temporal que suponía reentrenar un modelo ya preentrenado y por qué se supuso que, entrenar un *Word embedding* únicamente con los discursos contenidos en las actas llevaría, a una mejor representación de las relaciones semánticas que les dan los políticos dentro del congreso a los conceptos que usan y no tanto a una representación semántica general que podría tener el resto de la población (en este caso, los usuarios de la red social Twitter).

7.2 Modelo Deep Learning

En un primer momento, se llevaron a cabo una serie de pruebas mediante la librería SpaCy y su analizador de sentimientos, pero dado que esta librería utiliza para el entrenamiento *datasets* de *tweets* o comentarios en páginas en ingles habría que traducir los discursos lo que lleva a una bajada de precisión. Además, el conjunto mediante el que se entrena el analizador de sentimientos de SpaCy utiliza comentarios de distintos contextos ajenos al tratado en este trabajo como críticas de cine, de productos…

Algunos ejemplos de los resultados que arrojó la librería son los siguientes:

“Solo por eso debería cesar hoy mismo a su vicepresidente y debería también romper con Bildu, en vez de sacar la apisonadora de la propaganda el mismo día que acercan al asesino de nuestro compañero Goyo Ordóñez.”

SpaCy: Sentiment(polarity=-0.05, subjectivity=0.1125)

Se puede observar que, pese a ser un comentario polarizado, ofrece un valor muy bajo de polarización negativa (5%).

O, por ejemplo, este otro análisis:

"Señoría, lo que habría que preguntar es qué hacen ustedes. Ustedes, señoría, lo que han hecho miércoles tras miércoles ha sido levantarse, hacer aquí una perorata de improperios y de exabruptos, mezclados a veces con insultos."

SpaCy: Sentiment(polarity=0.0, subjectivity=0.25)

Donde ni siquiera detecta niveles de polarización, aunque sí de subjetividad. Por ello, se decidió construir un modelo de *Deep Learning* formado por LSTM Bidireccionales.

Una vez construido y entrenado con un subconjunto de todos los *tweets* disponibles (en concreto, se utilizó únicamente el *dataset* "strompol-train-tagged" para el entenamiento y "strompol-test-tagged" como conjunto de test dado que era el único que contenía en su totalidad *tweets* sobre política) el modelo LSTM entrenado con 580 *tweets* y construido mediante una capa oculta LSTM Bidireccional de 128 neuronas, alcanzó una precisión máxima del 53.8% y loss: 1.9, mientras que en las pruebas con una clasificación a 2 clases ("P" o "N") alcanzó 61% de precisión y *loss*: 0.6709. Este último modelo tenía una capa oculta de 128 neuronas y una capa de salida *Dense* con 1 neurona y activación sigmoide, dichos resultados de muestran en la **Tabla 17**.

Tabla 17: Comparación de los dos primeros modelos generados.

	Precisión	**Loss**
CategoricalCrossentropy 128 neuronas capa oculta	53.8%	1.9
BinaryCrossentropy 128 neuronas capa oculta	61%	0.6709

Al aumentar el número de *tweets* en nuestro *dataset* mediante el uso del resto de archivos XML, la precisión en el conjunto de test aumentó hasta el 90.53% con una *loss* de 0.3603.

Para el conjunto de actas que pasaremos a analizar, calculamos el número de intervenciones que realizó cada político en la **Tabla 18**:

Tabla 18: Intervenciones por diputado.

Político	**Intervenciones**
SANCHEZ	1873
CASADO	711
IGLESIAS	536

ABASCAL	358
ARRIMADAS	319

Como vemos, quien tiene mayor cantidad de intervenciones es Pedro Sánchez, esto es debido a que parte del periodo durante el que se recogen las actas Pedro Sánchez era candidato a la presidencia del gobierno por lo que en las sesiones de investidura es quien contaba con mayor número de intervenciones. Le sigue Pablo Casado como líder de la oposición y Pablo Iglesias como vicepresidente del gobierno. Finalmente se encuentran Santiago Abascal e Inés Arrimadas.

Para el cálculo de los valores de polaridad de los datos de entrada (en este caso, los discursos de las actas), se utilizó la función `get_sentiment`, que convierte el conjunto de ficheros de texto formados por los discursos de cada acta y los carga en memoria convirtiéndolos a objetos de tipo *Datasets* numéricos para su uso en el modelo como hemos realizado con el conjunto de entrenamiento y de *test*. Una vez hecho esto, llamamos a la función *predict* con el *dataset* como entrada del modelo. La función devuelve un tensor *predictions*, con los valores de polaridad calculados por el modelo por cada discurso y el objeto *dataset*.

En primer lugar, se muestra una tabla comparativa en la **Figura 17** entre los representantes de los cinco grandes partidos. Siendo *mean_P*, la media de la polaridad positiva, *mean_NEU* la media de polaridades neutrales, y *mean_N* las negativas, de cada diputado en todas las actas.

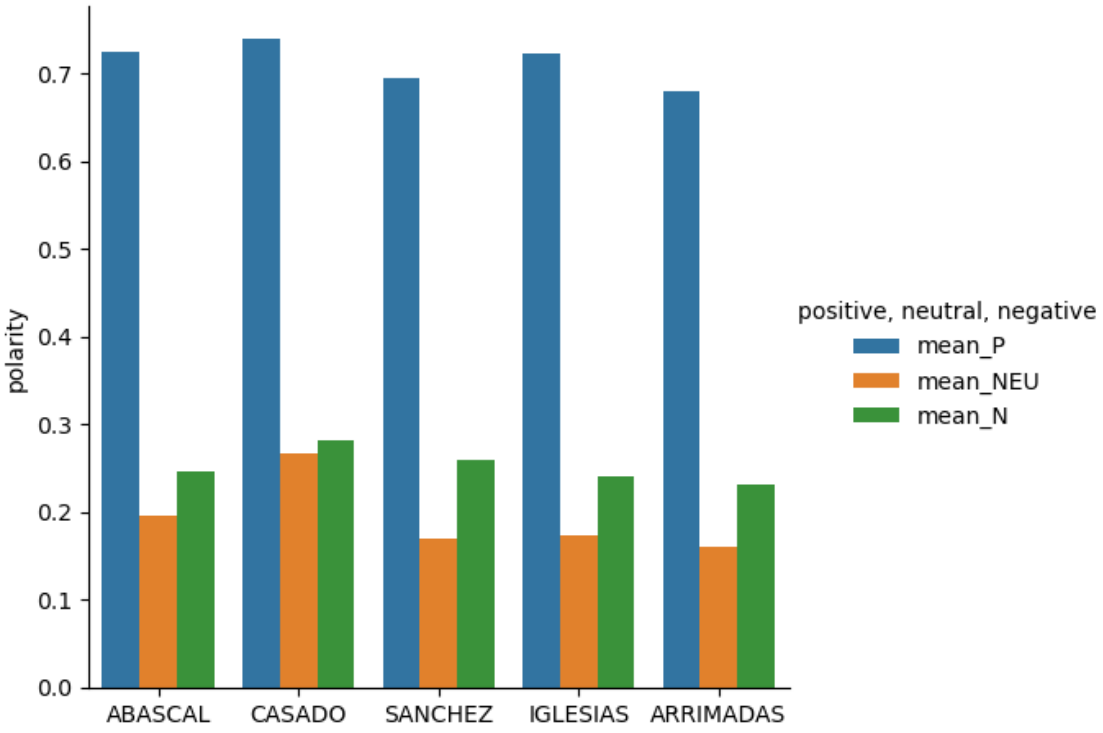


Figura 17: Grafica comparativa entre presidentes de partido del sentimiento.

Como se observa en la **Tabla 19**, el diputado con mayor polaridad media de sentimiento positivo ha sido Pablo Casado con un porcentaje de activación de la neurona que representa el sentimiento positivo del 0,74 de media. A continuación, estaría Santiago Abascal con 0,72. En tercer lugar, estaría Pablo Iglesias con un 0,71 y finalmente Pedro Sánchez e Inés Arrimadas con 0,695 y 0,678 respectivamente.

Para la activación de la neurona neutral, tenemos en los dos primeros lugares a Pablo Casado con 0.26 y Santiago Abascal con 0.195 respectivamente. A continuación, vendrían Pablo Iglesias con 0.17 y Pedro Sánchez con 0.16 y finalmente Inés Arrimadas seria la que ha causado menos activaciones de la neurona neutral con 0.15.

En la media de la neurona negativa tenemos en primer lugar a Pablo Casado con 0.28 siguiéndole Pedro Sánchez con 0.26. Después tenemos a Santiago Abascal con 0.25 y Pablo Iglesias con un 0.24. Finalmente, Inés Arrimadas con un 0.22.

Tabla 19: Porcentaje en tanto por uno de activación media de cada neurona para cada discurso de cada diputado.

Político	Media P	Media NEU	Media N
ABASCAL	0.725	0.196	0.247
CASADO	0.737	0.263	0.280
SANCHEZ	0.695	0.169	0.257
ARRIMADAS	0.701	0.090	0.208
IGLESIAS	0.678	0.161	0.230

En las siguientes graficas se observa la polarización obtenida a lo largo del tiempo correspondiente a la fecha del acta que se estuviera analizando. Cada grafica hace referencia a la activación de una de las tres neuronas de las que disponía nuestro modelo. En primer lugar, en la **Figura 18** se muestra la tendencia de la neurona positiva:

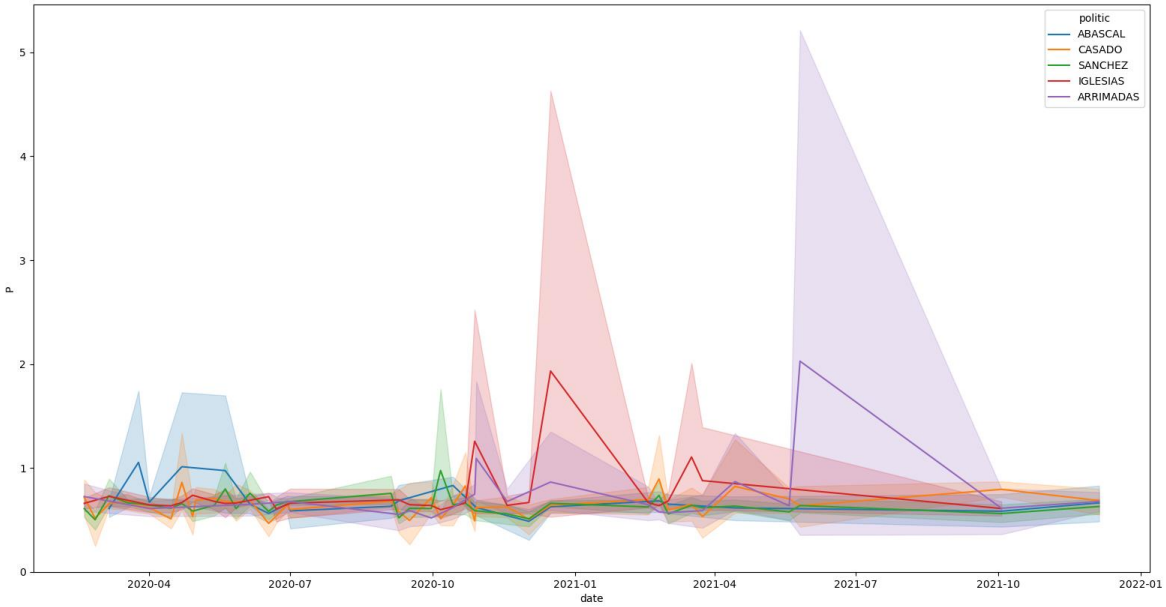


Figura 18: Polarización positiva a lo largo del tiempo de los diputados.

Sorprende el aumento de la media de polarización positiva en el mes de enero de 2021 por parte de Pablo Iglesias, ello puede ser debido a la aprobación en el congreso de los presupuestos generales del estado [45]. Además de ello, se tiene que en el mes de Junio Arrimadas presentó mayores activaciones para la neurona positiva.

Sobre todo, Abascal, aunque también Casado muestran mayores sentimientos positivos que el resto durante el periodo de abril hasta junio de 2020. Pedro Sánchez también muestra activaciones de esta neurona sobre julio de 2020 pudiendo deberse al anuncio de la llegada de la "nueva normalidad" [46].

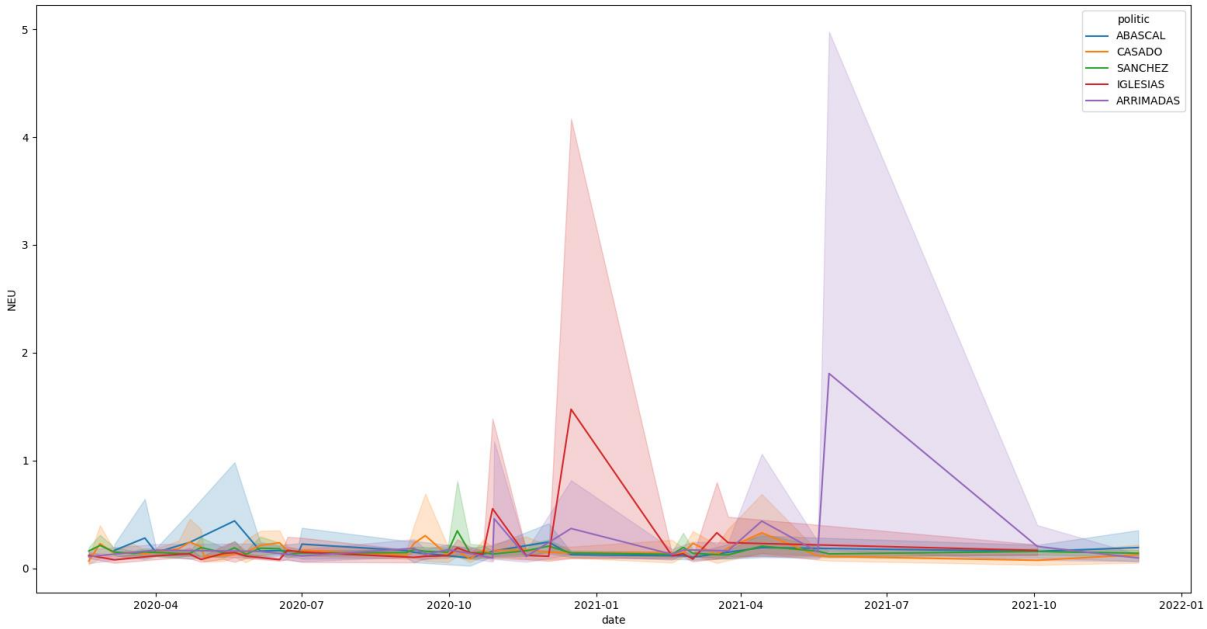


Figura 19: Polarización neutral a lo largo del tiempo de los diputados.

Se puede observaren la **Figura 19** como las tendencias son similares que con el grafico para la neurona positiva, predominando en un principio VOX (desde Marzo de 2020 hasta Junio de 2020 aproximadamente) para posteriormente pasar a una serie de picos de distintos partidos como el PP (en Octubre de 2020), el PSOE (en Noviembre de 2020) y tanto Iglesias como Arrimadas en Diciembre de 2020 para finalmente predominar Ciudadanos.

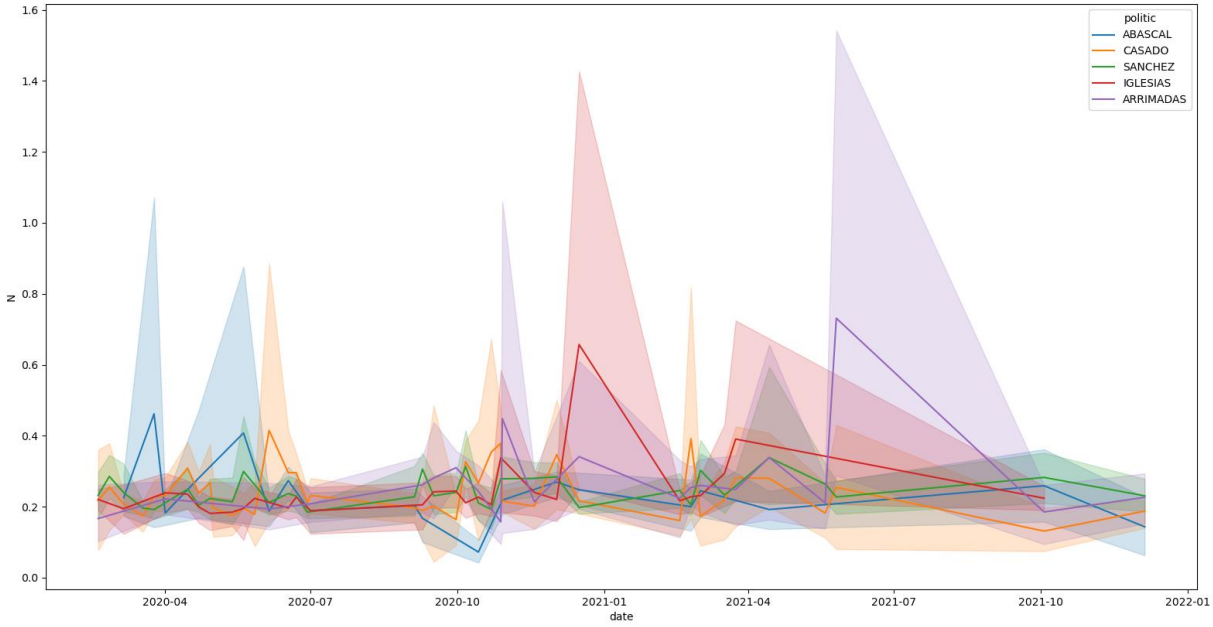


Figura 20: Polarización negativa a lo largo del tiempo de los diputados.

En la **Figura 20**, se pueden observar dos picos entre Marzo y Junio de 2020 del PP seguido de otro de Ciudadanos de polarización negativa en Junio de 2020. Posteriormente, el que genera más activaciones negativas es Pedro Sánchez junto con algunos picos de Inés Arrimadas y Santiago Abascal.

8. Conclusiones

El objetivo principal de este trabajo era realizar un análisis de sentimientos sobre las actas del congreso de los diputados. Para ello, se propuso el uso de modelos de redes neuronales conocidos, por ejemplo, un modelo de aprendizaje automático (*Deep Learning*). También se planteó el uso de un *Word Embedding* entrenado sobre los textos de las actas para determinar si se conseguían deducir relaciones semánticas o sintácticas. Ambos objetivos se han realizado y se han concluido.

Los problemas encontrados han sido principalmente de tipo técnicos. En un principio la velocidad de entrenamiento del modelo LSTM resultaba demasiado lenta debido a que cada entrada debía ser procesada individualmente convirtiéndola a un tensor numpy para ser introducido en el modelo. Esto se solucionó primero cargando en memoria el *dataset* entero formado por los ficheros de texto y convirtiéndolo directamente entero a un objeto de la clase *Dataset* de la librería Keras. Posteriormente la precisión obtenida era demasiado baja debido al número insuficiente de datos de entrenamiento por lo que se amplió el *dataset* hasta 7058 *tweets* de entrenamiento con lo que se consiguió una precisión suficiente sobre el conjunto de test.

Durante el desarrollo de este, se adquirió conocimiento en el uso de diversas librerías Python como keras o pandas.

8.1 Relación del trabajo desarrollado con los estudios cursados

Se escogió el lenguaje de programación Python presentado en **SAR** (Sistemas de almacenamiento y recuperación de datos) y **ALG** (Algoritmica), debido a que es un lenguaje que dispone de herramientas suficientes ya incluidas, así como también en forma de librerías en el ámbito del *machine learning* y del *Deep Learning*. Además, posee librerías para otras tareas que se requerían, tales como el preprocesado de los *tweets* y las actas.

En el campo de la inteligencia artificial, este proyecto ha ayudado a consolidar conocimientos de redes neuronales vistos en **APR** (Aprendizaje Automático).

8.2 Trabajos Futuros

Como posibles mejoras al trabajo realizado se encuentran las siguientes:

- Añadir un análisis de sentimiento a nivel de aspecto.
- Incorporar un detector de discurso de odio.

- Incorporar la detección de ironía o sarcasmo.

- Confeccionar un *dataset* etiquetado por expertos con los discursos de políticos en español de España para una mejora de la precisión.

- El uso de redes neuronales con Transformers los cuales han demostrado tener una gran utilidad práctica como en el modelo GPT-3 [24].

Bibliografía

[1] ZHANG, LEI, WANG, SHUAI and LIU, BING, 2018, Deep learning for sentiment analysis: A survey. *WIREs Data Mining and Knowledge Discovery* [online]. 2018. Vol. 8, no. 4. [Accessed 6 July 2021]. DOI 10.1002/widm.1253. Available from: https://doi.org/10.1002/widm.1253Wiley

[2] ARAQUE, OSCAR, CORCUERA-PLATAS, IGNACIO, SÁNCHEZ-RADA, J. FERNANDO and IGLESIAS, CARLOS A., 2017, Enhancing deep learning sentiment analysis with ensemble techniques in social applications. Expert Systems with Applications [online]. 2017. Vol. 77, p. 236-246. [Accessed 6 July 2021]. DOI 10.1016/j.eswa.2017.02.002. Available from: https://doi.org/10.1016/j.eswa.2017.02.002Elsevier BV

[3] TANG, DUYU, QIN, BING and LIU, TING, 2015, Deep learning for sentiment analysis: successful approaches and future challenges. WIREs Data Mining and Knowledge Discovery [online]. 2015. Vol. 5, no. 6, p. 292-303. [Accessed 6 July 2021]. DOI 10.1002/widm.1171. Available from: https://doi.org/10.1002/widm.1171Wiley

[4] AIN, Qurat Tul, et al. Sentiment analysis using deep learning techniques: a review. Int J Adv Comput Sci Appl, 2017, vol. 8, no 6.

[5] YADAV, ASHIMA and VISHWAKARMA, DINESH KUMAR, 2019, Sentiment analysis using deep learning architectures: a review. Artificial Intelligence Review [online]. 2019. Vol. 53, no. 6, p. 4335-4385. [Accessed 6 July 2021]. DOI 10.1007/s10462-019-09794-5. Available from: https://doi.org/10.1007/s10462-019-09794-5Springer Science and Business Media LLC

[6] DANG, NHAN CACH, MORENO-GARCÍA, MARÍA N. and DE LA PRIETA, FERNANDO, 2020, Sentiment Analysis Based on Deep Learning: A Comparative Study. Electronics [online]. 2020. Vol. 9, no. 3, p. 483. [Accessed 6 July 2021]. DOI 10.3390/electronics9030483. Available from: https://doi.org/10.3390/electronics9030483MDPI AG

[7] BENGIO, Yoshua, et al. A neural probabilistic language model. The journal of machine learning research, 2003, vol. 3.

[8] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. 2013, arXiv preprint arXiv:1301.3781.

[9] MIKOLOV, Tomas, et al. Distributed representations of words and phrases and their compositionality. En Advances in neural information processing systems. 2013.

[10] LIU, Bing. Sentiment analysis and opinion mining. Synthesis lectures on human language technologies, 2012, vol. 5, no 1, p. 1-167.

[11] RHEAULT, LUDOVIC, BEELEN, KASPAR, COCHRANE, CHRISTOPHER and HIRST, GRAEME, 2016, Measuring Emotion in Parliamentary Debates with Automated Textual Analysis. PLOS ONE [online]. 2016. Vol. 11, no. 12, p. e0168843. [Accessed 6 July 2021]. DOI 10.1371/journal.pone.0168843. Available from: https://doi.org/10.1371/journal.pone.0168843Public Library of Science (PLoS)

[12] EFRON, Miles. Cultural Orientation: Classifying Subjective Documents by Cociation Analysis. En AAAI Technical Report (7). 2004.

[13] GOLDBERG, Andrew B.; ZHU, Xiaojin; WRIGHT, Stephen. Dissimilarity in graph-based semi-supervised classification. En Artificial Intelligence and Statistics. PMLR, 2007.

[14] GREFENSTETTE, Gregory, et al. Coupling niche browsers and affect analysis for an opinion mining application. Proceedings of Recherche d'Information Assistée par Ordinateur (RIAO), 2004.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, Efficient Estimation of Word Representations in Vector Space, Arxiv, no. 9, pp. 112, 2013.

[16] BOJANOWSKI, PIOTR, GRAVE, EDOUARD, JOULIN, ARMAND and MIKOLOV, TOMAS, 2017, Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics [online]. 2017. Vol. 5, p. 135-146. [Accessed 6 July 2021]. DOI 10.1162/tacl_a_00051. Available from: https://doi.org/10.1162/tacl_a_00051MIT Press - Journals

[17] P. Bojanowski*, E. Grave*, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information

[18] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification

[19] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, T. Mikolov, FastText.zip: Compressing text classification models

[20] LIWICKI, Marcus, et al. Combining diverse systems for handwritten text line recognition. Machine vision and applications, 2011, vol. 22, no 1, p. 39-51.

[21] TensorBoard, TensorFlow, [en linea]. [consulta: 1 de junio de 2021] Disponible en: https://www.tensorflow.org/tensorboard/graphs

[22] FOLGADO, Miguel G.; SANZ, Veronica. Exploring the political pulse of a country using data science tools. arXiv preprint arXiv:2011.10264, 2020.

[23] HUTTO, Clayton; GILBERT, Eric. Vader: A parsimonious rule-based model for sentiment analysis of social media text. En Proceedings of the International AAAI Conference on Web and Social Media. 2014.

[24] BROWN, Tom B., et al. Language models are few-shot learners. arXiv

preprint arXiv:2005.14165, 2020.

[28] TASS, SEPLN [en línea], @2015 [consulta: 1 de junio de 2021] Disponible en: http://tass.sepln.org/2015/tass2015.php

[29] LIU, Bing; ZHANG, Lei. A survey of opinion mining and sentiment analysis. En Mining text data. Springer, Boston, MA, 2012. p. 415-463.

[30] CHOWDHURY, GOBINDA G., 2005, Natural language processing. Annual Review of Information Science and Technology [online]. 2005. Vol. 37, no. 1, p. 51-89. [Accessed 6 July 2021]. DOI 10.1002/aris.1440370103. Available from: https://doi.org/10.1002/aris.1440370103Wiley

[31] PEDREGOSA, Fabian, et al. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 2011, vol. 12, p. 2825-2830.

[32] SpaCy [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://spacy.io/

[34] Python urllib website. https://docs.python.org/3/library/urllib.html

[35] Python Beautiful Soup [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[36] Python Gensim FastText Model [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://radimrehurek.com/gensim/models/fasttext.html

[37] Keras [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://keras.io/

[38] FastText [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://fasttext.cc/docs/en/faqs.html

[39] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[40] Proceedings of the TASS workshop at SEPLN 2013. Actas del XXIX Congreso de la Sociedad Española de Procesamiento de Lenguaje Natural. IV Congreso Español de Informática. 17-20 September 2013, Madrid, Spain. Díaz Esteban, Alberto; Alegría, Iñaki; Villena Román, Julio (eds). ISBN: 978-84-695-8349-4. http://www.congresocedi.es/images/site/actas/ActasSEPLN.pdf.

Villena-Román, Julio, Lana-Serrano, Sara, Martínez-Cámara, Eugenio, González-Cristobal, José Carlos. 2013. TASS (2012) - Workshop on Sentiment Analysis at SEPLN. Revista de Procesamiento del Lenguaje Natural, 50, pp 37-44. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4657.

TASS (Taller de Análisis de Sentimientos en la SEPLN) [en linea]. [consulta: 1 de julio de 2021]. Disponible en: http://www.daedalus.es/TASS.

[41] HUNTER, JOHN D., 2007, Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering [online]. 2007. Vol. 9, no. 3, p. 90-95. [Accessed 6 July 2021]. DOI 10.1109/mcse.2007.55. Available from: http://10.1109/MCSE.2007.55Institute of Electrical and Electronics Engineers (IEEE)

[42] REBACK, JEFF, MCKINNEY, WES, BOSSCHE, JORIS, AUGSPURGER, TOM, CLOUD, PHILLIP, KLEIN, ADAM, ROESCHKE, MATTHEW, HAWKINS, SIMON, TRATNER, JEFF, SHE, CHANG, AYD, WILLIAM, PETERSEN, TERJI, GARCIA, MARC, SCHENDEL, JEREMY, HAYDEN, ANDY, JANCAUSKAS, VYTAUTAS, BATTISTON, PIETRO, SEABOLD, SKIPPER, HOYER, STEPHAN, OVERMEIRE, WOUTER, DONG, KAIQI, WHELAN, CHRISTOPHER and MEHYAR, MORTADA, 2021, pandas-dev/pandas: Pandas 1.0.3. Zenodo [online]. 2021. [Accessed 6 July 2021]. Available from: http://doi.org/10.5281/zenodo.3715232

[43] Python documentation webpage for xml.dom.minidom [en linea]. [consulta: 1 de julio de 2021]. Disponible en: https://docs.python.org/3/library/xml.dom.minidom.html

[44] WASKOM, MICHAEL, 2021, seaborn: statistical data visualization. Journal of Open Source Software [online]. 2021. Vol. 6, no. 60, p. 3021. [Accessed 6 July 2021]. DOI 10.21105/joss.03021. Available from: http://10.21105/joss.03021The Open Journal

[45] HERMIDA, XOSÉ and CASQUEIRO, JAVIER, 2021, Los Presupuestos del Gobierno de coalición logran un amplio aval del Congreso. elpais.es [online]. 2021. [Accessed 7 July 2021]. Available from: https://elpais.com/espana/2020-12-03/los-presupuestos-del-gobierno-de-coalicion-logran-un-amplio-aval-del-congreso.html

[46] LUGILDE, ANXO, 2021, Sánchez pide no “bajar la guardia” ante los rebrotes y “disfrutar de la nueva normalidad”. La Vanguardia [online]. 2021. [Accessed 4 July 2021]. Available from: https://www.lavanguardia.com/politica/20200704/482089574834/pedro-sanchez-virus-derribar-gobierno-elecciones-gallegas-12j.html