Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Análisis de emociones en textos escritos mediante técnicas de aprendizaje automático

Autora: Desirée García Soriano
Tutor: Rubén Martín Clemente

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Análisis de emociones en textos escritos mediante técnicas de aprendizaje automático

Autora:
Desirée García Soriano

Tutor:
Rubén Martín Clemente
Profesor titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Máster: Análisis de emociones en textos escritos mediante técnicas de aprendizaje automático

Autora: Desirée García Soriano

Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

Agradecimientos

Termino otra etapa en mi vida y quería agradecer primeramente a mi familia, sobre todo a mis padres y a mi hermana, por animarme siempre y apoyarme en los momentos más difíciles.

A Julio, por acompañarme y ayudarme siempre que he necesitado ayuda.

Gracias a mis amigos y compañeros que he conocido en el máster, con los cuales he pasado muchas horas de estudio y trabajo en la escuela y con los que me he reído tanto.

Por último, agradecer a mi tutor Rubén por orientarme en este trabajo y a los demás profesores y maestros, de los que he aprendido en toda mi etapa educativa.

Desirée García Soriano

Sevilla, 2021

Resumen

El Procesamiento del Lenguaje Natural es la disciplina que estudia cómo hacer que las máquinas lean e interpreten el lenguaje que utilizan las personas, el lenguaje natural. Sin embargo, en el mundo de las máquinas no existen las palabras como tal, sólo existen secuencias de números utilizadas para representar caracteres con el fin de mostrarlos en una pantalla. Una de las ramas del Procesamiento del Lenguaje Natural es el Análisis de Sentimientos, una tarea llevada a cabo por una máquina que se encarga de analizar y predecir a partir de una frase o texto el sentimiento o la opinión que le produciría a una persona al leerlo. En este documento realizaremos un estudio comparativo de algunos de los algoritmos de aprendizaje automático (Machine Learning) más utilizados, tales como "Support Vector Machines" y "Random Forest", para llevar a cabo un análisis de emociones en tweets tanto en inglés como en español.

Abstract

Natural Language Processing is the discipline that studies how to make machines read and interpret the language that people use, natural language. However, in the machine world there are no words as such, there are only sequences of numbers used to represent characters in order to display them on a screen. One of the branches of Natural Language Processing is Sentiment Analysis, a task carried out by a machine that is in charge of analyzing and predicting from a sentence or text the feeling or opinion that a person would have when reading it. In this paper we will make a comparative study of some of the most used Machine Learning algorithms, such as "Support Vector Machines" and "Random Forest", to carry out an analysis of emotions in tweets in both English and Spanish.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xvii**

Índice de Figuras **xix**

Índice de Códigos **xxi**

1 Introducción **1**
1.1 Motivación del trabajo *1*
1.2 Objetivos y enfoque *2*
1.3 Estructura del documento *2*

2 Machine Learning para Análisis de Sentimientos **3**
2.1 Procesamiento del lenguaje natural *3*
2.2 Análisis de sentimientos en textos *4*
2.3 Clasificación automática de textos *4*
2.3.1 Procesos de entrenamiento y predicción 5
2.3.2 Extracción de características del texto 5
2.3.3 Algoritmos de clasificación 9
2.4 Librerías y software existentes *15*

3 Metodología **17**
3.1 Procedimiento seguido *17*
3.2 Obtención de datos *18*
3.2.1 Tweets en inglés 18
3.2.2 Tweets en español 20
3.3 Preprocesamiento del texto *21*
3.3.1 Tweets en inglés 21
3.3.2 Tweets en español 28
3.4 Vectores de palabras *30*
3.4.1 Experimentación de ejemplo 31
3.4.2 Vectores de tweets 36
3.5 Implementación de clasificadores y experimentación *38*
3.5.1 Experimentación previa 39
3.5.2 Experimentación 41
3.5.3 GridSearchCV 45

4 Evaluación de Resultados **47**
4.1 Métricas de evaluación *47*
4.2 Evaluación de los modelos SVM *49*
4.2.1 Idioma inglés 49
4.2.2 Idioma español 51
4.3 Evaluación de los modelos Random Forest *53*

4.3.1 Idioma inglés 53
4.3.2 Idioma español 55
4.4 Muestra de tweets clasificados 56
5 Conclusiones y Líneas de Continuación 59
Referencias 61

ÍNDICE DE TABLAS

Tabla 2-1. Ejemplo de datos de entrenamiento para el modelo *Skip-Gram* de *Word2Vec*. 7

Tabla 3-1. Valores de la métrica "exactitud" en experimentación previa con el doble de datos. 40

Tabla 3-2. Valores de la métrica "exactitud" en experimentación previa con conjunto *stemmatizado* en inglés. 41

Tabla 3-3. Valores de la métrica "exactitud" en experimentación previa con conjunto *stemmatizado* en español. 41

Tabla 3-4. Valores de los hiperparámetros con los que se experimentará en *SVM*. 44

Tabla 3-5. Fórmulas matemáticas del hiperparámetro *criterion* de *Random Forest* para medir la calidad de una división. 44

Tabla 3-6. Valores de los hiperparámetros con los que se experimentará en *Random Forest*. 45

Tabla 4-1. Resultados experimentación *SVM* en el idioma inglés. 50

Tabla 4-2. Resultados experimentación *SVM* en el idioma español. 52

Tabla 4-3. Resultados experimentación *Random Forest* en el idioma inglés. 54

Tabla 4-4. Resultados experimentación *Random Forest* en el idioma español. 55

ÍNDICE DE FIGURAS

Figura 1-1. Análisis de Sentimientos en el Procesamiento del Lenguaje Natural. 1
Figura 2-1. Clasificación automática de textos. 4
Figura 2-2. Ejemplo de *bag-of-words*. 5
Figura 2-3. Ejemplo de *Word Embeddings*. 6
Figura 2-4. Arquitectura de la red neuronal para el modelo *Skip-Gram* de *Word2Vec*. 8
Figura 2-5. Cálculo matricial. 8
Figura 2-6. Arquitectura del modelo *Doc2Vec*. 9
Figura 2-7. *Support Vector Machines*. 10
Figura 2-8. Márgenes en *SVM*. 10
Figura 2-9. Datos desordenados. 11
Figura 2-10. Hiperplano en el caso de tres dimensiones. 11
Figura 2-11. Árbol de decisión. 12
Figura 2-12. Creación de conjuntos de datos individuales. 13
Figura 2-13. Selección aleatoria de características para construir el árbol nº1. 13
Figura 2-14. Clasificación usando *Random Forest*. 14
Figura 3-1. Esquema de la metodología utilizada. 17
Figura 3-2. Muestra del formato de tweet en inglés. 18
Figura 3-3. Muestra de tweets en inglés barajados. 19
Figura 3-4. Diagrama de caja de tweets en inglés. 20
Figura 3-5. Muestra del formato de tweet en español. 20
Figura 3-6. Muestra de tweets en español barajados. 21
Figura 3-7. Diagrama de caja de tweets en español. 21
Figura 3-8. Muestra de tweets en inglés (Figura 3-3) tras haber sido preprocesados con la librería *re*. 24
Figura 3-9. Muestra de palabras *tokenizadas* de los tweets en inglés (Figura 3-8). 25
Figura 3-10. Muestra de tweets en inglés (Figura 3-8) tras realizar corrección ortográfica, eliminación de *stopwords* y *stemming*. 26
Figura 3-11. Nubes de palabras de tweets en inglés con polaridad negativa (izquierda) y polaridad positiva (derecha). 27
Figura 3-12. Muestra de tweets en español (Figura 3-6) tras haber sido preprocesados con la librería *re*. 28
Figura 3-13. Muestra de tweets en español (Figura 3-12) tras realizar eliminación de *stopwords* y *stemming*. 29
Figura 3-14. Nubes de palabras de tweets en español con polaridad negativa (izquierda) y polaridad positiva (derecha). 30
Figura 3-15. Palabras más similares a “car”. 31
Figura 3-16. Palabras más similares a “cake”. 32

Figura 3-17. Palabras más similares a “amazing”. 32
Figura 3-18. Similitud entre las palabras “computer” y “laptop”. 32
Figura 3-19. Similitud entre las palabras “computer” y “phone”. 33
Figura 3-20. Similitud entre las palabas “boy” y “plane”. 33
Figura 3-21. Palabra sobrante entre “bottle”, “glass” y “sun”. 33
Figura 3-22. Palabra sobrante entre “spoon”, “light” y “fork”. 33
Figura 3-23. Analogía de palabras entre “woman”, “boy” y “man”. 34
Figura 3-24. Analogía de palabras entre “canada”, “paris” y “france”. 34
Figura 3-25. Palabras más similares a otras relacionadas con comida. 35
Figura 3-26. Representación en dos dimensiones de vectores de palabras con *t-SNE*. 36
Figura 3-27. Muestra de tweets etiquetados. 37
Figura 3-28. Matriz con vectores de tweets. 38
Figura 3-29. Validación cruzada. 42
Figura 3-30. Influencia del hiperparámetro *C* en el algoritmo *SVM*. 43
Figura 3-31. Influencia del hiperparámetro *kernel* en el algoritmo *SVM*. 43
Figura 3-32. Influencia del hiperparámetro *gamma* en el algoritmo *SVM*. 44
Figura 4-1. Estructura de la matriz de confusión. 48
Figura 4-2. Exactitud del mejor modelo de *SVM* con los datos de test del idioma inglés. 50
Figura 4-3. Matriz de confusión del mejor modelo de *SVM* con los datos de test del idioma inglés. 51
Figura 4-4. Exactitud del mejor modelo de *SVM* con los datos de test del idioma español. 52
Figura 4-5. Matriz de confusión del mejor modelo de *SVM* con los datos de test del idioma español. 53
Figura 4-6. Exactitud del mejor modelo de *Random Forest* con los datos de test del idioma inglés. 54
Figura 4-7. Matriz de confusión del mejor modelo de *Random Forest* con los datos de test del idioma inglés. 54
Figura 4-8. Exactitud del mejor modelo de *Random Forest* con los datos de test del idioma español. 55
Figura 4-9. Matriz de confusión del mejor modelo de *Random Forest* con los datos de test del idioma español. 56
Figura 4-10. Muestra de tweets correctamente clasificados. 56
Figura 4-11. Muestra de tweets incorrectamente clasificados. 57

ÍNDICE DE CÓDIGOS

Código 3-1. Método *read_csv()*. 18
Código 3-2. Método *drop()* y barajamiento de tweets. 19
Código 3-3. Representación del diagrama de caja de la longitud de los tweets. 19
Código 3-4. Ejemplo de patrones de la librería *re* (I). 22
Código 3-5. Ejemplo de patrones de la librería *re* (II). 22
Código 3-6. Ejemplo de patrones de la librería *re* (III). 22
Código 3-7. Función *re_cleaner()* en el caso inglés. 23
Código 3-8. *Tokenización* de palabras. 24
Código 3-9. Corrección de palabras. 25
Código 3-10. Eliminación de *stopwords* en inglés. 25
Código 3-11. Realización de *stemming* en inglés. 26
Código 3-12. Representación de *WordCloud*. 27
Código 3-13. Eliminación de tildes. 28
Código 3-14. Eliminación de *stopwords* en español. 28
Código 3-15. Realización de *stemming* en español. 29
Código 3-16. Creación y entrenamiento del modelo *Word2Vec*. 31
Código 3-17. Algoritmo *t-SNE*. 35
Código 3-18. Creación del modelo y realización de la media de vectores de palabras con *Word2Vec*. 37
Código 3-19. Etiquetado de tweets. 37
Código 3-20. Creación del modelo y vectores de tweets con *Doc2Vec*. 38
Código 3-21. Algoritmo *SVM*. 39
Código 3-22. Algoritmo *Random Forest*. 40
Código 3-23. *GridSearchCV* para *SVM*. 45
Código 4-1. Métricas *Accuracy*, *F1-score* y *ROC-AUC*. 48
Código 4-2. Matriz de confusión. 49

1 INTRODUCCIÓN

1.1 Motivación del trabajo

La aparición de las redes sociales y la fácil disponibilidad del acceso a Internet, han hecho que cada vez más personas compartan sus impresiones y sentimientos de productos, empresas, series, películas o lo que deseen a través de la red. Twitter [1] y Facebook [2] son dos claros ejemplos de redes donde la gente describe cómo se siente hoy o si el producto que ha adquirido en una tienda es bueno o no. Por lo tanto, la capacidad de analizar y procesar esta información se ha convertido en algo importante puesto que, si introducimos un nuevo producto en el mercado, podemos esperar a ver la reacción de los consumidores en Internet, extraer estos sentimientos y decidir la viabilidad futura de este nuevo producto.

El principal problema que existe es que la mayor parte de esta información no está clasificada y es complicado clasificar de forma masiva todos estos datos manualmente o con herramientas que no estén preparadas para realizar esta tarea. Es por ello por lo que el desarrollo de herramientas que sean capaces de leer textos y asignarles a los mismos sentimientos es importante para el futuro.

El Procesamiento del Lenguaje Natural (NLP) es la rama de la inteligencia artificial que se ocupa de dotar a las máquinas de la capacidad de comprender textos de forma muy similar a la de las personas. Dentro de esta rama se encuentra el Análisis de Sentimientos, el cual estudia cómo utilizar las máquinas para procesar textos y asociar a cada uno de ellos una emoción. Esta rama utiliza algoritmos para el procesamiento del lenguaje con el fin de extraer características, como la frecuencia de palabras, y algoritmos de aprendizaje que aprenden a partir de un conjunto de datos inicial clasificado manualmente [3].

En este trabajo se llevará a cabo un análisis de sentimientos usando, para ello, dos conjuntos de datos con tweets en inglés y en español, y se utilizarán diversas técnicas de aprendizaje automático que trabajan de manera diferente y, como consecuencia, aprenden de distintas formas con los mismos datos.

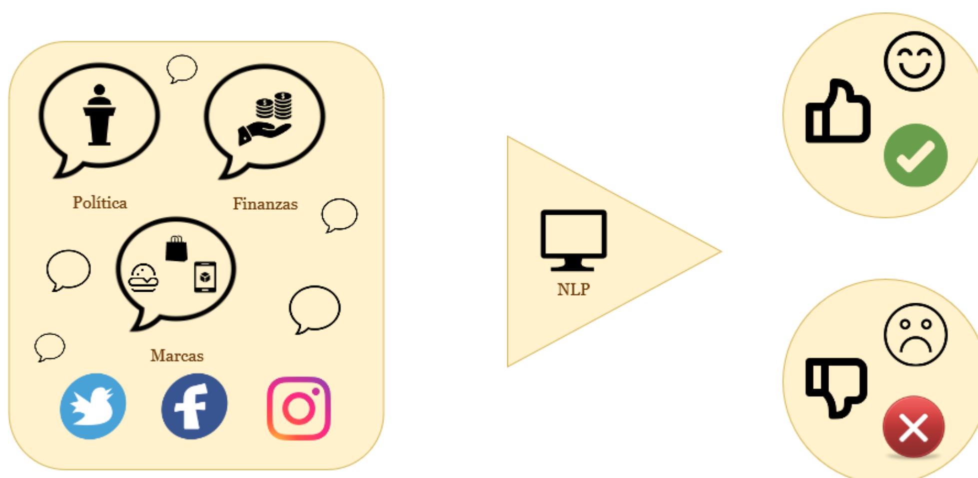

Figura 1-1. Análisis de Sentimientos en el Procesamiento del Lenguaje Natural.

1.2 Objetivos y enfoque

El objetivo de este trabajo es realizar el estudio, implementación y evaluación de diferentes métodos de aprendizaje automático (Machine Learning) utilizados para el análisis de sentimientos en textos escritos, concretamente en tweets en los idiomas inglés y español.

Dado que los tweets no están escritos todos de la misma forma (palabras en mayúsculas y minúsculas, faltas de ortografía, tildes, etc) se procederá inicialmente con una limpieza y normalización de todos ellos, además de una eliminación de palabras sin significado y extracción de raíces de palabras.

Por otro lado, puesto que los algoritmos de aprendizaje automático no reciben palabras como tal, sino vectores de números, de alguna forma debemos realizar una transformación previa sobre las mismas, extrayendo sus características.

1.3 Estructura del documento

El documento está estructurado en 5 bloques, organizados de la siguiente forma:

- **Capítulo 1: Introducción.** Se describe el contexto del trabajo, sus objetivos y la estructura del documento.
- **Capítulo 2: Machine Learning para Análisis de Sentimientos.** Se explica en primer lugar en qué consiste el Procesamiento del Lenguaje Natural y su rama sobre el Análisis de Sentimientos. Tras ello, se explica la clasificación automática de textos y se termina mostrando las librerías y el software existente para llevarla a cabo.
- **Capítulo 3: Metodología.** Se procede a describir la metodología a llevar a cabo y se detalla cómo se han realizado el preprocesamiento del texto, la extracción de características, la implementación de los clasificadores y los experimentos que se van a realizar, indicando en estos últimos los hiperparámetros de los algoritmos que irán variando en cada uno de ellos.
- **Capítulo 4: Evaluación de Resultados.** Se evalúa el rendimiento que ofrece cada uno de los modelos que se han implementado, mostrando los valores que se han obtenido para diferentes métricas.
- **Capítulo 5: Conclusiones y Líneas de Continuación.** Se finaliza con las conclusiones obtenidas como resultado del trabajo y posibles mejoras de éste.

2 Machine Learning para Análisis de Sentimientos

2.1 Procesamiento del lenguaje natural

El procesamiento del lenguaje natural (NLP) se refiere a la rama de la inteligencia artificial [4] que se ocupa de dotar a los ordenadores de la capacidad de comprender textos y palabras habladas como hacen los seres humanos.

El NLP combina la lingüística computacional (el modelado del lenguaje humano basado en reglas) con modelos estadísticos, de aprendizaje automático y de aprendizaje profundo. Juntas, estas tecnologías permiten a los ordenadores procesar el lenguaje humano en forma de texto o datos de voz y "entender" su significado completo, con la intención y el sentimiento del hablante o escritor.

El NLP permite que los programas informáticos traduzcan textos de un idioma a otro, respondan a órdenes habladas y resuman grandes volúmenes de texto con rapidez, incluso en tiempo real. Es muy probable que en algún momento de nuestras vidas hayamos interactuado con el NLP en forma de sistemas de GPS operados por voz, asistentes digitales, software de dictado de voz a texto, chatbots de atención al cliente y otras comodidades para el consumidor. Pero el NLP también desempeña un papel cada vez más importante en las soluciones empresariales que ayudan a agilizar las operaciones de la empresa, a aumentar la productividad de los empleados y a simplificar los procesos empresariales críticos.

El lenguaje humano está lleno de ambigüedades que hacen increíblemente difícil escribir un software que determine con precisión el significado de un texto o de los datos de voz. Homónimos, homófonos, sarcasmo, modismos, metáforas, excepciones gramaticales y de uso, variaciones en la estructura de las frases... son sólo algunas de las irregularidades del lenguaje humano que los humanos tardan años en aprender, pero que los programadores deben enseñar a las aplicaciones basadas en el lenguaje natural a reconocer y entender con precisión desde el principio, si quieren que esas aplicaciones sean útiles.

El procesamiento del lenguaje natural es el motor de la inteligencia artificial en muchas aplicaciones modernas del mundo real. Un ejemplo de ello es el análisis de sentimientos en las redes sociales, en el cual nos vamos a centrar en este trabajo.

El NLP se ha convertido en una herramienta empresarial esencial para descubrir los datos ocultos de los canales de las redes sociales. El análisis de sentimientos puede analizar el lenguaje utilizado en las publicaciones de las redes sociales, las respuestas, los comentarios, etc., para extraer actitudes y emociones en respuesta a los productos, las promociones y los eventos, información que las empresas pueden utilizar en el diseño de productos, las campañas publicitarias, etc [5].

2.2 Análisis de sentimientos en textos

El análisis de sentimientos es el proceso de detección de sentimientos positivos o negativos en un texto. Las empresas lo utilizan a menudo para detectar el sentimiento en las redes sociales, medir la reputación de la marca y comprender a los clientes.

Dado que los clientes expresan sus pensamientos y sentimientos de forma más abierta que nunca, el análisis de sentimientos se está convirtiendo en una herramienta esencial para controlar y comprender ese sentimiento. El análisis automático de los comentarios de los clientes, como las opiniones en las respuestas a las encuestas y las conversaciones en las redes sociales, permite a las marcas saber qué hace que los clientes estén contentos o frustrados, para poder adaptar los productos y servicios a las necesidades de sus clientes [6].

Por ejemplo, utilizar el análisis de sentimientos para analizar automáticamente más de 4.000 opiniones sobre un producto podría ayudar a descubrir si los clientes están contentos con las tarifas y el servicio de atención al cliente, o tal vez se quiera medir el sentimiento de la marca en las redes sociales, en tiempo real y a lo largo del tiempo, para poder detectar inmediatamente a los clientes descontentos y responder lo antes posible.

El análisis de sentimientos, también conocido como minería de opinión, funciona gracias al procesamiento del lenguaje natural y a los algoritmos de aprendizaje automático, para determinar automáticamente el tono emocional de las conversaciones online [7].

2.3 Clasificación automática de textos

Un problema de análisis de sentimientos se suele modelar como un problema de clasificación, en el que un clasificador recibe un texto y devuelve una categoría, por ejemplo, positiva o negativa.

En un clasificador de aprendizaje automático existen dos procesos a seguir para su implementación tal y como se muestra en la Figura 2-1:

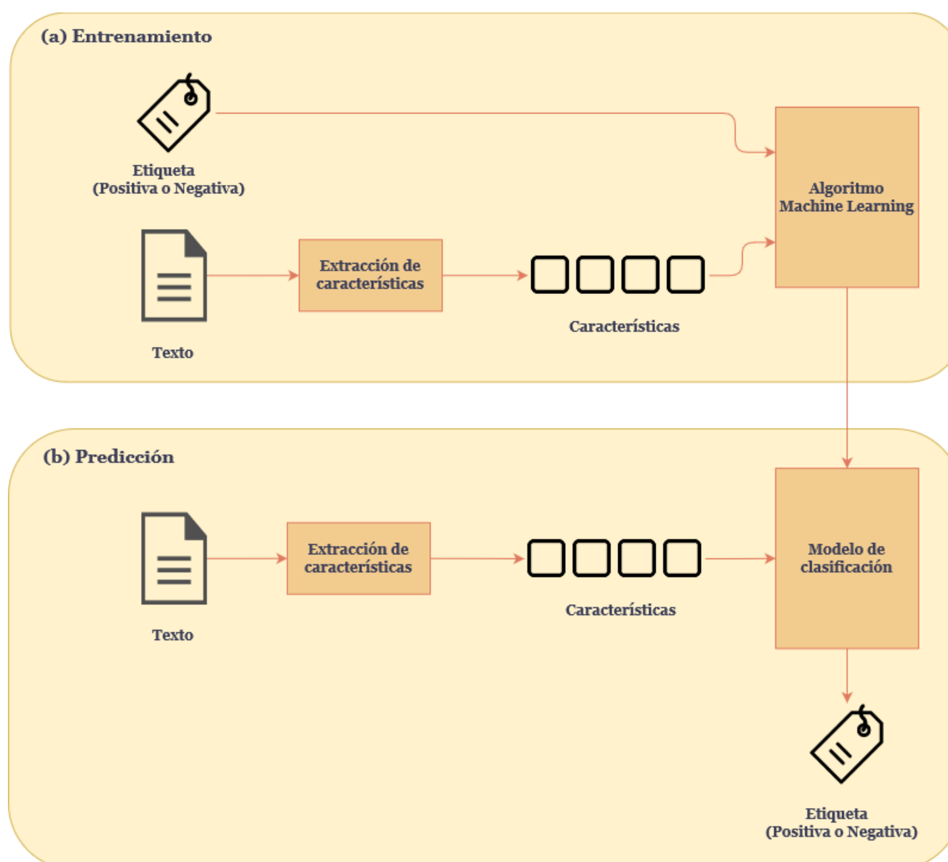

Figura 2-1. Clasificación automática de textos.

2.3.1 Procesos de entrenamiento y predicción

En el proceso de entrenamiento (Figura 2-1 (a)), nuestro modelo aprende a asociar una entrada concreta (es decir, un texto) a la salida correspondiente (etiqueta) basándose en las muestras de prueba utilizadas para el entrenamiento. El extractor de características convierte la entrada de texto en un vector de características. Los pares de vectores de características y etiquetas (por ejemplo, positiva o negativa) se introducen en el algoritmo de aprendizaje automático para generar un modelo.

En el proceso de predicción (Figura 2-1 (b)), el extractor de características se utiliza para transformar las entradas de texto no vistas en vectores de características. Estos vectores de características se introducen en el modelo, que genera las etiquetas predichas (de nuevo, positivas o negativas) [7].

2.3.2 Extracción de características del texto

El primer paso en un clasificador de texto de aprendizaje automático es realizar la extracción de las características del texto o la vectorización del mismo, y el enfoque clásico ha sido el de *bag-of-words* (bolsa de palabras).

El modelo *bag-of-words* es un método que se utiliza en el procesamiento del lenguaje para representar textos ignorando el orden de las palabras. La idea detrás de este modelo es muy simple y nos permite representar texto a través de un vector de características numéricas [8]. Los pasos para su construcción son:

1. Crear un vocabulario de palabras únicas para todo el conjunto total de textos que tengamos.
2. Para cada uno de los textos, se construye un vector de características que contiene el número de veces que cada palabra del vocabulario aparece dentro de él.

Por ejemplo, si tenemos los siguientes tres textos:

- The cat sat
- The cat sat in the hat
- The cat with the hat

El vocabulario resultante de los mismos sería: “the”, “cat”, “sat”, “in”, “hat”, “with”, y la frase “the cat sat” se representaría como el vector de características (the: 1, cat: 1, sat: 1, in: 0, hat: 0, with: 0) = (1, 1, 1, 0, 0, 0).

En la Figura 2-2 se puede ver cómo quedarían los vectores de características de los tres textos:

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

Figura 2-2. Ejemplo de *bag-of-words*.

Cabe destacar que la obtención de los vectores de características del modelo *bag-of-words* se realiza de forma automática, sin intervención humana, a través de librerías existentes de programación.

Como principales problemas que se encuentran al utilizar este método de extracción de características es que no se tiene en cuenta el contexto de las palabras en las frases y, por otro lado, cada vector contiene todas las palabras existentes en el conjunto total, con lo cual puede suponer un problema en el rendimiento de computación.

Más recientemente, se han aplicado nuevas técnicas de extracción de características basadas en incrustaciones de palabras (también conocidas como *Word Embeddings*). La premisa en la que se basan estas técnicas es que es posible caracterizar una palabra por el resto de palabras de las que aparezca acompañada. Al igual que en el modelo de *bag-of-words*, también se construyen vectores de características numéricas. Sin embargo, en este caso, estos vectores están formados por números reales, los cuales representan coordenadas en un determinado espacio vectorial. Gracias a ellos podemos calcular palabras próximas, o análogas, en función de la distancia que exista entre sus vectores. Es decir, cuanto más cerca se encuentren dos vectores de palabras, éstas tendrán una semántica más similar que de encontrarse más alejados. El cálculo de esta similitud o cercanía entre vectores puede hacerse de diversas maneras, siendo algunas de las más utilizadas la "distancia euclídea" o la "similitud coseno" [9]. Al igual que en el caso de *bag-of-words*, esta asignación de palabras a vectores se realiza de forma automática a través de librerías de programación existentes.

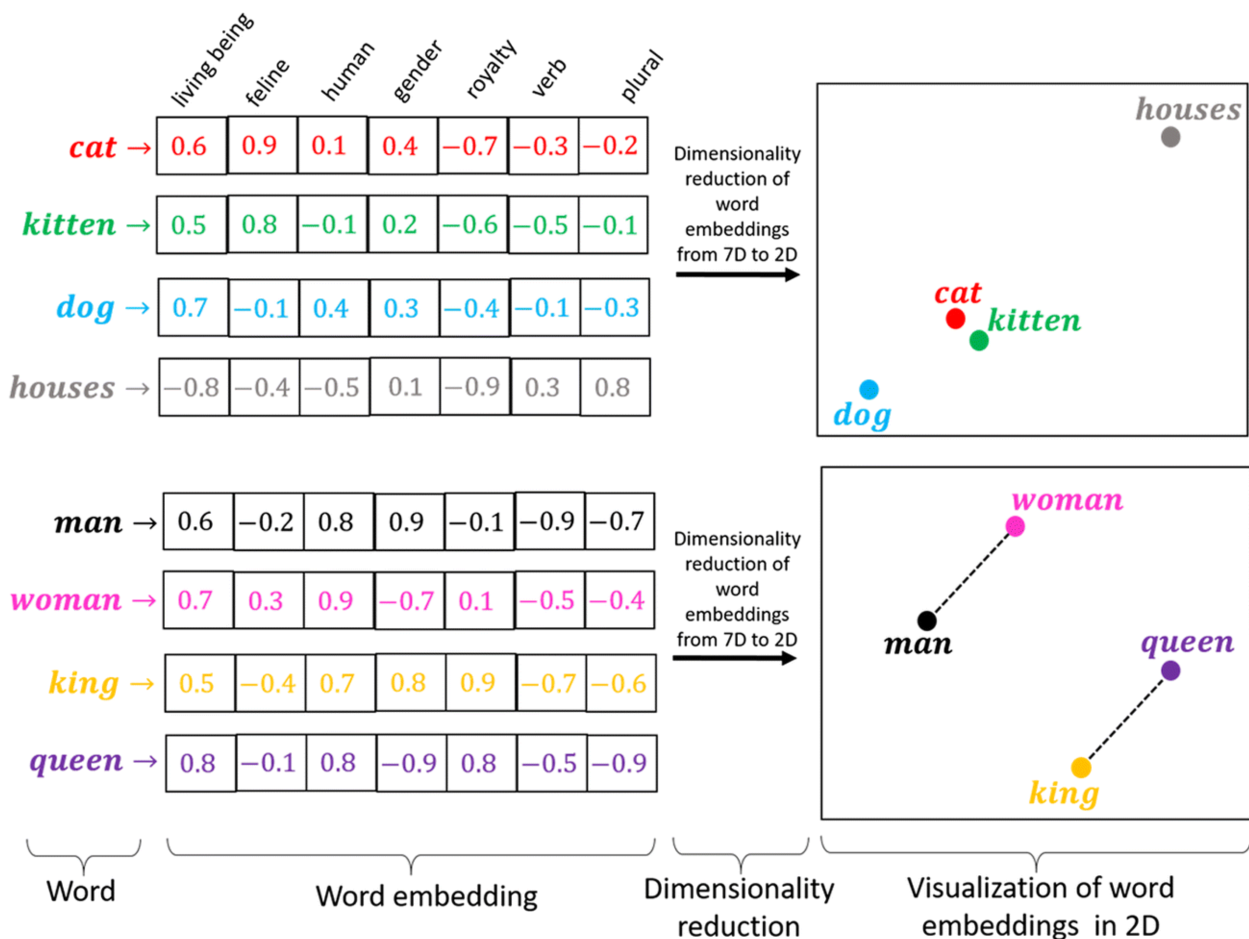

Figura 2-3. Ejemplo de *Word Embeddings*.

En la Figura 2-3, podemos observar un ejemplo de vectores de características obtenidos a través de *Word Embeddings*. En la parte superior tenemos cuatro vectores que representan las palabras "cat" (gato), "kitten" (gatito), "dog" (perro) y "houses" (casas). Por otro lado, tenemos siete dimensiones y cada una de ellas representa una categoría: "living being" (ser vivo), "feline" (felino), "human" (humano), "gender" (género), "royalty" (realeza), "verb" (verbo) y "plural" (plural). Para cualquiera de los cuatro vectores, el valor que tiene la coordenada correspondiente a una dimensión es una medida del grado de pertenencia de la palabra a esa categoría. Por ejemplo, las palabras "cat" y "kitten" tienen un valor alto en la categoría "feline" pero "dog" y "houses" tienen un valor bajo en la misma.

Si representamos estos vectores en dos dimensiones, observamos que las palabras "cat" y "kitten" están muy cercanas entre sí, ya que ambas tienen significados muy similares. La palabra "dog" está un poco más alejada de ellas porque, aunque no tiene tanta similitud con las anteriores, sí corresponde a un ser vivo animal. Sin

embargo, “houses” no tiene similitud con ellas y aparece representada muy alejada del resto.

En la parte inferior de la imagen están representados otros cuatro vectores, cada uno con sus correspondientes coordenadas. Al realizar la representación de los mismos en dos dimensiones, observamos que la distancia y dirección existentes entre las palabras “man” (hombre) y “woman” (“mujer”) son las mismas que entre “king” (rey) y “queen” (reina). Es decir, el modelo nos está indicando que existe una relación entre “man” y “woman” que es muy similar a la que hay entre “king” y “queen”.

2.3.2.1 Word2Vec

Uno de los modelos de *Word Embeddings* existentes es *Word2Vec. Word2Vec* utiliza una red neuronal simple con una sola capa oculta para aprender los pesos. A diferencia de la mayoría de los modelos de aprendizaje automático, no nos interesan las predicciones que pueda hacer esta red neuronal. En su lugar, sólo nos importan los pesos de la capa oculta, ya que estos pesos son en realidad las incrustaciones/vectores de palabras que vamos a aprender.

Word2Vec se compone de dos modelos de aprendizaje diferentes: *Continuous Bag-of-Words model (CBOW)* y *Skip-Gram*.

Para el modelo *Skip-Gram*, la tarea de la red neuronal es: dada una palabra de entrada en una frase, la red predecirá la probabilidad de que cada palabra del vocabulario sea una palabra “cercana” a esa palabra de entrada.

Los datos de entrenamiento de la red neuronal son pares de palabras que consisten en la palabra de entrada y sus palabras “cercanas”. Por ejemplo, consideremos la frase "He says make America great again" y un tamaño de ventana de 2. Los datos de entrenamiento son los siguientes:

Sentence	Training examples
He says make America great again.	(he,says), (he,make)
He **says** make America great again.	(says,he), (says,make), (says,america)
He says **make** America great again.	(make,he), (make,says), (make,america), (make,great)
He says make **America** great again.	(america,says),(america,make) (america,great),(america,again)
He says make America **great** again.	(great,make), (great,america),(great,again)
He says make America great **again**.	(again,america), (again,great)

Tabla 2-1. Ejemplo de datos de entrenamiento para el modelo *Skip-Gram* de *Word2Vec*.

Para que los datos puedan ser entrenados por la red neuronal, hay que representar las palabras de alguna forma numérica. Se utilizan vectores *one-hot*, en los que la posición de la palabra de entrada es "1" y todas las demás posiciones son "0". Por tanto, las entradas de la red neuronal son simplemente vectores de entrada *one-hot*, y la salida es también un vector con la dimensión del vector *one-hot*, que contiene, para cada palabra del vocabulario, la probabilidad de que una palabra “cercana” seleccionada al azar sea esa palabra del vocabulario.

Como ejemplo, supongamos que utilizamos un vocabulario de tamaño V y una capa oculta de tamaño N. La siguiente figura muestra la arquitectura de la red:

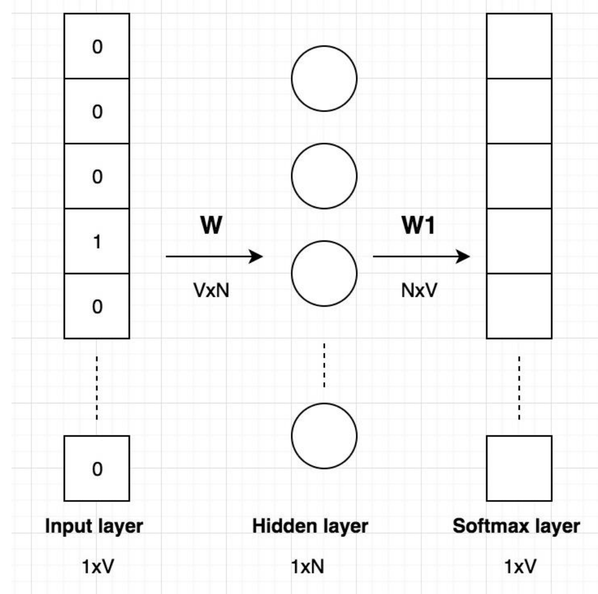

Figura 2-4. Arquitectura de la red neuronal para el modelo *Skip-Gram* de *Word2Vec*.

La entrada es un vector *one-hot* con una dimensión 1xV. La dimensión de la matriz de pesos de la capa oculta es VxN. Si los multiplicamos, obtendremos un vector de dimensión 1xN.

$$[0\;0\;0\;0\;1\;0] \times \begin{bmatrix} 10 & 23 & 15 \\ 3 & 14 & 9 \\ 18 & 26 & 2 \\ 10 & 17 & 7 \\ 12 & 23 & 8 \\ 9 & 10 & 12 \end{bmatrix} = [12\;\;23\;\;8]$$

Figura 2-5. Cálculo matricial.

Si se observa realmente el cálculo matricial de la Figura 2-5, se puede ver que la matriz de pesos de la capa oculta funciona en realidad como una tabla de búsqueda, que efectivamente sólo seleccionará la fila de la matriz correspondiente al "1". La salida del cálculo de la matriz es el vector de incrustación de la palabra de entrada. Hay V filas en la matriz de pesos, cada una de las cuales corresponde a un vector de palabras del vocabulario. Por eso sólo nos interesa aprender la matriz de pesos de la capa oculta, la cual se conoce como *Word Embeddings*.

La capa de salida es una capa *softmax* con una dimensión 1xV, en la que cada elemento corresponde a la probabilidad de que esa palabra sea la que se selecciona aleatoriamente “cerca” de la palabra de entrada.

El caso de *CBOW* es justo lo contrario que el de *Skip-Gram*. La tarea de la red neuronal es: dado un contexto de palabras (que rodea a una palabra) en una frase, la red predecirá la probabilidad de que cada palabra del vocabulario sea esa palabra [10].

2.3.2.2 Doc2Vec

Doc2Vec es una extensión de *Word2Vec*, de manera que *Word2Vec* extrae vectores de características de palabras mientras que *Doc2Vec* extrae vectores de características de textos añadiendo, para ello, otro vector (ID de texto) a la entrada. La arquitectura del modelo *Doc2Vec* se muestra a continuación:

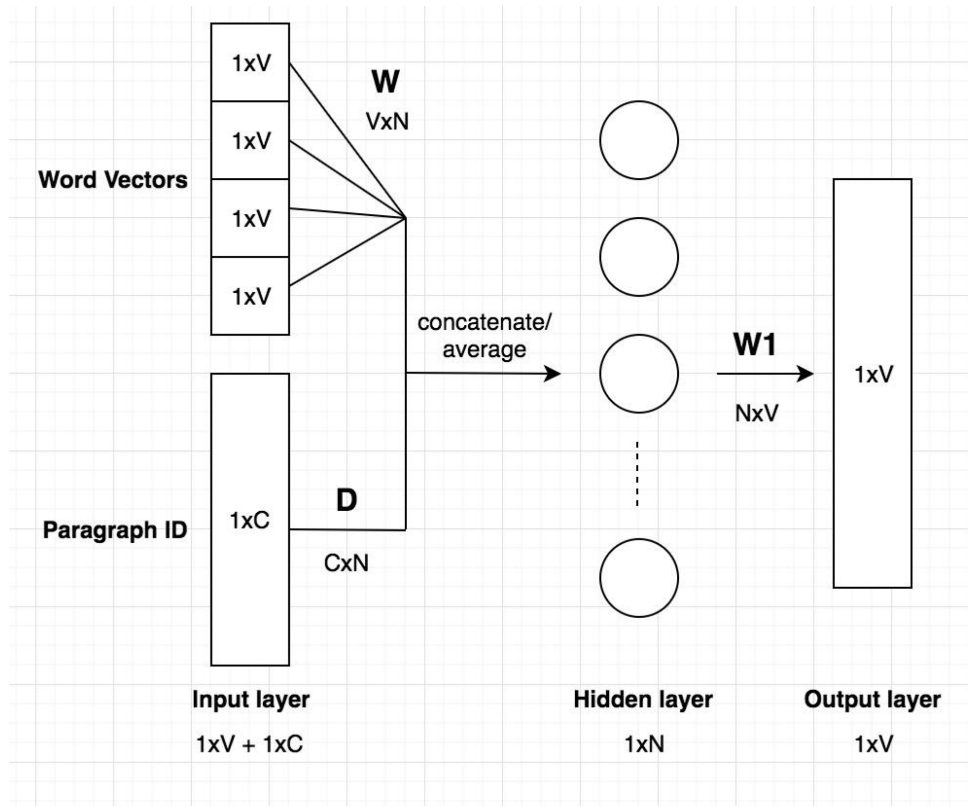

Figura 2-6. Arquitectura del modelo *Doc2Vec*.

El diagrama de la Figura 2-6 se basa en el modelo *CBOW*, pero en lugar de utilizar sólo las palabras "cercanas" para predecir la palabra, también se añade otro vector de características, que es único para el texto. Así, al entrenar los vectores de palabras *W*, se entrena también el vector de textos *D*, que al final del entrenamiento contiene una representación numérica del texto.

Las entradas consisten en vectores de palabras y vectores de identificación del texto. El vector de palabras es un vector *one-hot* con una dimensión 1xV. El vector *Id del texto* tiene una dimensión de 1xC, donde *C* es el número de textos totales. La dimensión de la matriz de pesos *W* de la capa oculta es VxN. La dimensión de la matriz de pesos *D* de la capa oculta es CxN.

El modelo anterior se denomina *Memory version of Paragraph Vector (PV-DM)*. Existe también otro algoritmo *Doc2Vec* que se basa en *Skip-Gram*, denominado *Distributed Bag of Words version of Paragraph Vector (PV-DBOW)* [10].

2.3.3 Algoritmos de clasificación

La etapa de clasificación suele implicar un modelo como *Support Vector Machines* y *Random Forest*, los dos algoritmos que se utilizarán en este trabajo.

2.3.3.1 Support Vector Machines

Una máquina de vectores de soporte (*SVM*) es un algoritmo de aprendizaje automático supervisado que puede emplearse tanto para la clasificación como para la regresión. Las *SVM* se utilizan más comúnmente en problemas de clasificación, como es en nuestro caso.

Las *SVM* se basan en la idea de encontrar un hiperplano que divida mejor un conjunto de datos en dos clases, como se muestra en la Figura 2-7:

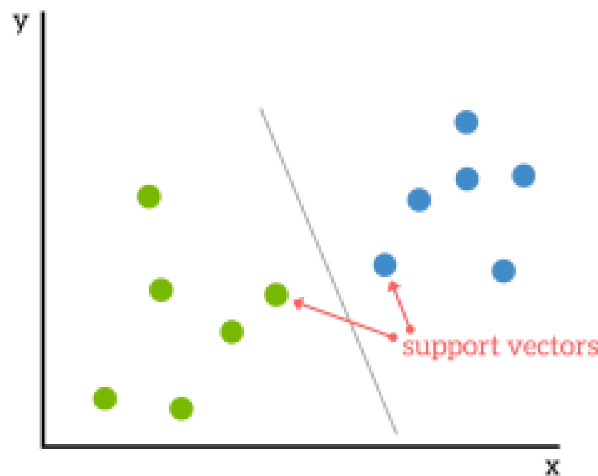

Figura 2-7. *Support Vector Machines*.

Los vectores de soporte son los puntos de datos más cercanos al hiperplano, los puntos de un conjunto de datos que, si se eliminaran, alterarían la posición del hiperplano de división. Por ello, pueden considerarse los elementos críticos de un conjunto de datos.

Como ejemplo sencillo, para una tarea de clasificación con sólo dos características (como en la Figura 2-8), se puede pensar en un hiperplano como una línea que separa y clasifica linealmente un conjunto de datos. Intuitivamente, cuanto más lejos del hiperplano se encuentren nuestros puntos de datos, más seguros estaremos de que han sido clasificados correctamente. Por lo tanto, queremos que nuestros puntos de datos estén lo más lejos posible del hiperplano, sin dejar de estar en el lado correcto del mismo. Así, cuando se añadan nuevos datos, el lado del hiperplano en el que caigan decidirá la clase a asignarles.

Para encontrar el hiperplano correcto, de modo que segregue lo mejor posible las dos clases, hay que definir lo que se conoce como “margen”. El margen es la distancia entre el hiperplano y el punto de datos más cercano a cualquiera de los dos conjuntos. El objetivo es elegir un hiperplano con el mayor margen posible entre él mismo y cualquier punto del conjunto de entrenamiento, lo que proporciona una mayor probabilidad de que los nuevos datos se clasifiquen correctamente.

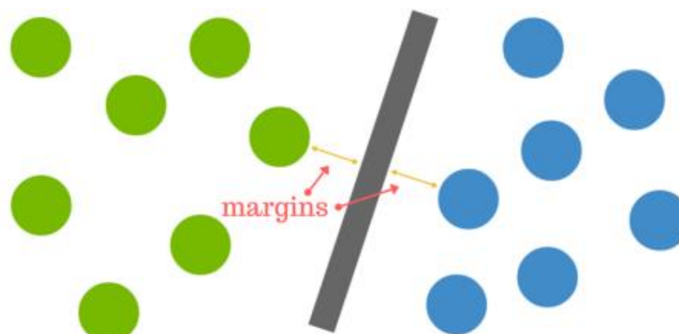

Figura 2-8. Márgenes en *SVM*.

Hay casos en los que resulta complicado encontrar un hiperplano de forma clara. Los datos rara vez son tan limpios como en el ejemplo anterior. Un conjunto de datos a menudo se parecerá más a las bolas desordenadas de la Figura 2-9, que representan un conjunto de datos linealmente no separable.

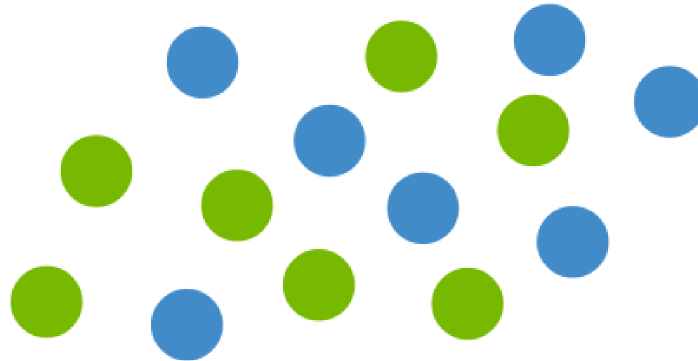

Figura 2-9. Datos desordenados.

Para clasificar un conjunto de datos como el anterior es necesario pasar de una visión 2D de los datos a una visión 3D. Por ejemplo, imaginemos que nuestros dos conjuntos de bolas de colores de la Figura 2-9 están puestos sobre una hoja y esta hoja se levanta de repente, lanzando las bolas al aire. Mientras las bolas están en el aire, podemos utilizar la hoja para separarlas entre sí. Este "levantamiento" de las bolas representa el mapeo de los datos en una dimensión superior. Esto se conoce como *kernelling*.

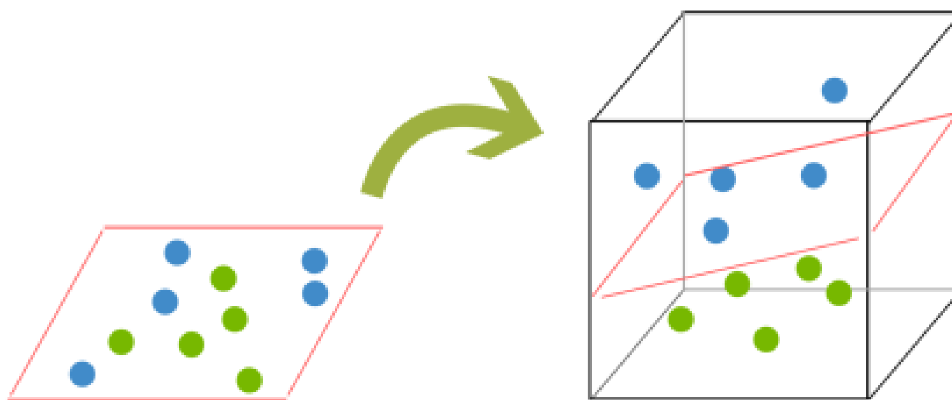

Figura 2-10. Hiperplano en el caso de tres dimensiones.

Como ahora estamos en tres dimensiones, nuestro hiperplano ya no puede ser una línea. Ahora debe ser un plano, como se muestra en la Figura 2-10. La idea es que los datos seguirán siendo mapeados en dimensiones cada vez más altas hasta que se pueda formar un hiperplano que los segregue [11].

2.3.3.2 Random Forest

Random Forest (Bosque Aleatorio) es un algoritmo de aprendizaje automático que puede utilizarse tanto para la regresión como para la clasificación. Es uno de los métodos de conjunto más populares, que pertenecen a la categoría específica de métodos "bagging".

Los métodos *ensemble* implican el uso de muchos aprendices para mejorar el rendimiento de cualquiera de ellos individualmente. Estos métodos pueden describirse como técnicas que utilizan un grupo de aprendices débiles de forma conjunta, con el fin de crear uno más fuerte y agregado.

En nuestro caso, los bosques aleatorios son un conjunto de muchos árboles de decisión individuales. Un árbol de decisión es un mapa de los posibles resultados de una serie de decisiones relacionadas. Comienza con un único nodo y luego se ramifica en resultados posibles. Cada uno de esos resultados crea nodos adicionales que se ramifican en otras posibilidades. Esto le da una forma similar a la de un árbol.

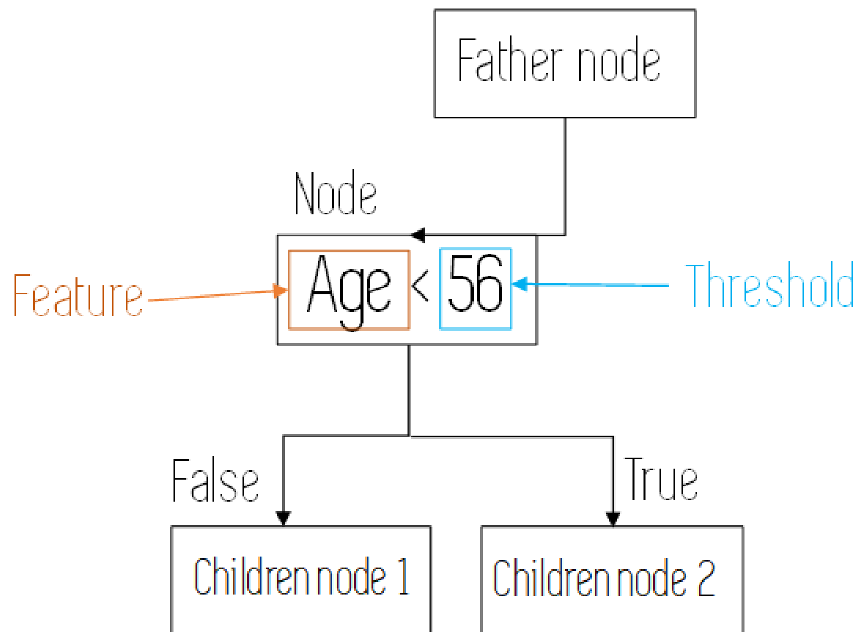

Figura 2-11. Árbol de decisión.

Uno de los principales inconvenientes de los árboles de decisión es que son muy propensos a sobreajustarse: funcionan bien con los datos de entrenamiento, pero no son tan flexibles para hacer predicciones sobre muestras no vistas. Los modelos de bosques aleatorios combinan la simplicidad de los árboles de decisión con la flexibilidad y la potencia de un modelo de conjunto. Aunque los modelos de bosques aleatorios no ofrecen tanta capacidad de interpretación como en el caso en el que sólo existe un solo árbol, su rendimiento es mucho mejor, y no tenemos que preocuparnos tanto de afinar perfectamente los parámetros del bosque como hacemos con los árboles individuales.

La construcción de un bosque aleatorio tiene 3 fases principales:

1. **Creación de un conjunto inicial de datos para cada árbol.**

   Cuando construimos un árbol de decisión individual, utilizamos un conjunto de datos de entrenamiento y todas las observaciones. Esto significa que, si no tenemos cuidado, el árbol puede ajustarse muy bien a estos datos de entrenamiento y generalizar mal a nuevas observaciones no vistas.

   Para construir un bosque aleatorio tenemos que entrenar N árboles de decisión y, para cada uno de ellos, elegiremos una muestra aleatoria de todo el conjunto de datos, como se muestra en la siguiente figura:

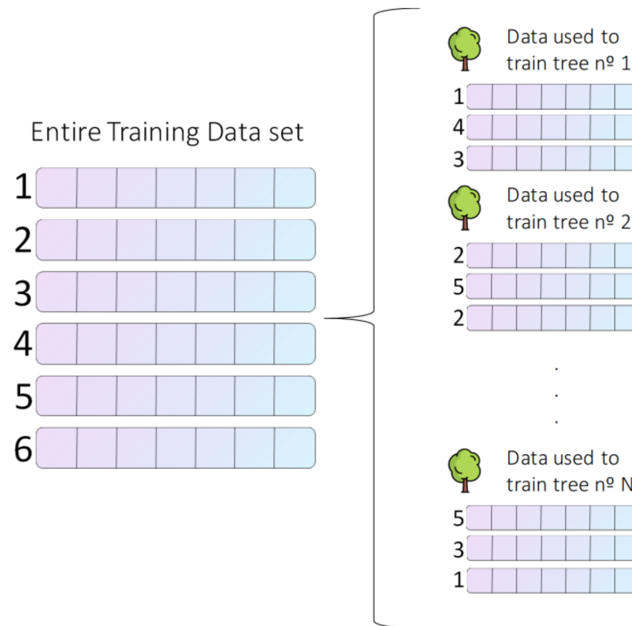

Figura 2-12. Creación de conjuntos de datos individuales.

En la Figura 2-12 podemos observar que el tamaño de los datos utilizados para entrenar cada árbol individual no tiene por qué ser el de todo el conjunto de datos. Además, un vector de datos puede estar presente más de una vez en los datos utilizados para entrenar un árbol (como ocurre en el árbol nº 2). Esto se denomina *muestreo con reemplazo* o *Bootstrapping*: cada vector de datos se elige al azar de todo el conjunto de datos, y un vector de datos puede elegirse más de una vez.

Al utilizar diferentes muestras de datos para entrenar cada árbol individual reducimos uno de los principales problemas que tienen: son muy afines a sus datos de entrenamiento. Si entrenamos un bosque con muchos árboles y cada uno de ellos ha sido entrenado con datos diferentes, solucionamos este problema. Todos son muy afines a sus datos de entrenamiento, pero el bosque no es afín a ningún vector de datos específico. Esto nos permite hacer crecer árboles individuales más grandes, puesto que ya no nos preocupa tanto que un árbol individual se sobreajuste.

2. **Entrenar el bosque de árboles utilizando conjuntos de datos aleatorios y selección de características aleatorias.**

   Para aumentar la aleatoriedad en el bosque de árboles, se seleccionan aleatoriamente ciertas características para evaluarlas en cada nodo.

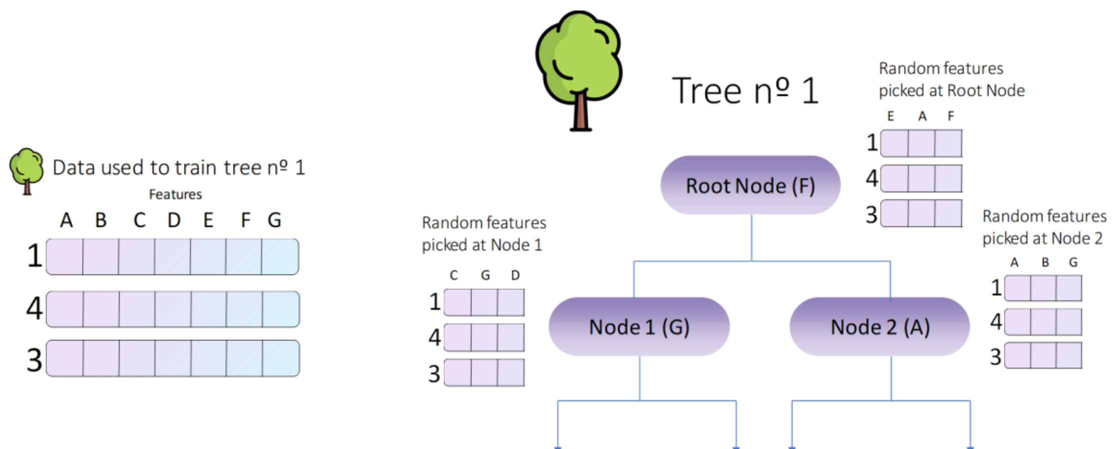

Figura 2-13. Selección aleatoria de características para construir el árbol nº1.

Como se puede ver en la Figura 2-13, en cada nodo evaluamos sólo un subconjunto de todas las características iniciales. En el Nodo Raíz tenemos en cuenta *E*, *A* y *F* (y gana *F*). En el Nodo 1 consideramos *C*, *G* y *D* (y gana *G*). Por último, en el Nodo 2 consideramos sólo *A*, *B* y *G* (y gana A). Seguiríamos haciendo esto hasta construir todo el árbol.

Al realizar este método, evitamos incluir características que tienen un poder predictivo muy alto en cada árbol, mientras que creamos muchos árboles no relacionados entre sí. Es decir, estamos aumentando la aleatoriedad. No sólo utilizamos datos aleatorios, sino también características aleatorias al construir cada árbol. Cuanto mayor sea la diversidad de árboles, mejor: reducimos la varianza y obtenemos un modelo de mejor rendimiento.

3. **Repetir el paso anterior con los N árboles para crear el bosque.**

   Una vez construido un único árbol de decisión, se repite lo mismo para los N árboles, seleccionando aleatoriamente en cada nodo de cada uno de los árboles qué variables entran en concurso para ser elegidas como característica a dividir.

Una vez se crea el bosque aleatorio al completo, se comienzan a realizar clasificaciones. Para ello, se toman cada uno de los árboles individuales, se les pasa la observación para la que queremos hacer una clasificación a través de ellos, se obtiene una clase de cada árbol (sumando hasta N clases) y luego se obtiene una clasificación global, agregada. La clase final es la más frecuente realizada por el bosque [12].

La Figura 2-14 ilustra este procedimiento con un ejemplo. Se quiere predecir si un determinado paciente está enfermo o sano. Para ello, pasamos su historial médico y otra información por cada árbol del bosque aleatorio, y obtenemos N predicciones (400 en este caso). En este ejemplo, 355 de los árboles dicen que el paciente está sano y 45 dicen que está enfermo, por lo que el bosque decide que el paciente está sano.

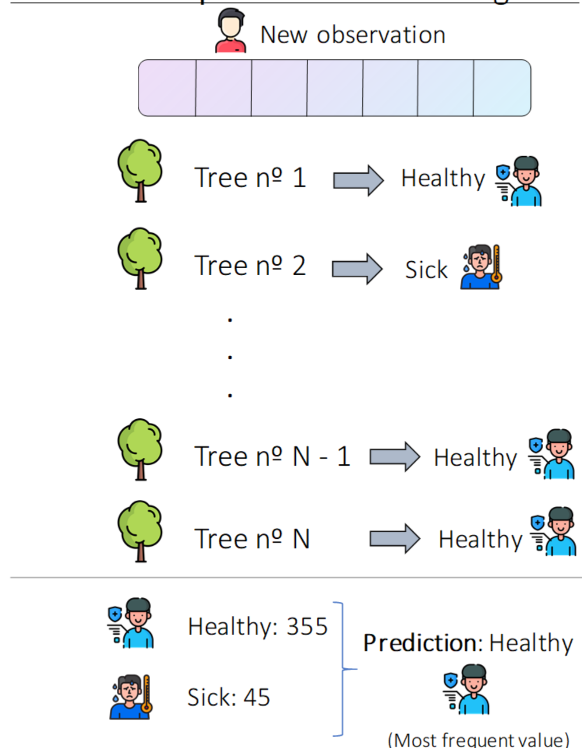

Figura 2-14. Clasificación usando *Random Forest*.

2.4 Librerías y software existentes

Actualmente, Python [13] es uno de los lenguajes más populares para trabajar en el campo de la inteligencia artificial [14]. Para resolver problemas relacionados con el Procesamiento del Lenguaje Natural, Python nos ofrece las siguientes bibliotecas:

- **NLTK:** es la principal librería para el Procesamiento del Lenguaje Natural. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos, junto con un conjunto de bibliotecas de procesamiento de texto para la clasificación, la *tokenización*, el *stemming*, el etiquetado, el análisis sintáctico y el razonamiento semántico [15].
- **RE:** las expresiones regulares (llamadas REs, o regexes, o patrones regex) son esencialmente un pequeño lenguaje de programación altamente especializado incrustado dentro de Python y disponible a través del módulo *re*. Usando este pequeño lenguaje, se pueden especificar reglas para cadenas que satisfagan preguntas como: "¿Coincide esta cadena con tal patrón?", o "¿Existe una coincidencia de este patrón en alguna parte de esta cadena?". De esta forma, podemos modificar cadenas que se encuentren dentro de un texto por cualquier otra diferente [16].
- **Beautiful Soup:** es una biblioteca de Python para analizar documentos HTML. Esta biblioteca crea un árbol con todos los elementos del documento y puede ser utilizado para extraer información. Por lo tanto, esta biblioteca es útil para realizar *web scraping* (extraer información de sitios web) [17].
- **SymSpell:** librería de Python para realizar correcciones ortográficas de palabras en inglés [18].
- **Unidecode:** librería que se utiliza para realizar la transformación de caracteres Unicode a caracteres ASCII. También se usa para eliminar las tildes en los caracteres [19].
- **Word Cloud:** biblioteca de Python para realizar nubes de palabras a partir de un texto [20].
- **Gensim:** biblioteca de código abierto para procesar textos, trabajar con modelos de vectores de palabras (como *Word2Vec*, *FastText*, etc.) y para construir modelos temáticos [21].
- **Scikit-learn:** biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de grupos entre los cuales están *máquinas de vectores de soporte*, *bosques aleatorios*, *Gradient boosting*, *K-means* y *DBSCAN* [22].

Por otra parte, para llevar a cabo la programación del código Python, es muy común utilizar los llamados “Notebooks” en Machine Learning. Se tratan de documentos que contienen tanto código (por ejemplo, Python) como elementos de texto enriquecido (párrafos, ecuaciones, figuras, enlaces, etc).

Existen dos interfaces muy utilizadas para la realización de Notebooks: *Jupyter Notebook* (de la distribución Anaconda) y *Google Colab*:

- **Jupyter Notebook:** interfaz web de código abierto que permite la inclusión de texto, vídeo, audio, imágenes, así como la ejecución de código a través del navegador en múltiples lenguajes. Esta ejecución se realiza mediante la comunicación con un núcleo (Kernel) de cálculo [23].
- **Google Colab:** servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google, con librerías como: *Scikit-learn*, *PyTorch*, *TensorFlow*, *Keras* y *OpenCV* [24].

3 METODOLOGÍA

Tenemos como objetivo principal implementar diferentes métodos de aprendizaje automático para realizar un análisis de sentimientos y clasificar tweets dependiendo de su polaridad. Esta clasificación se hará tanto para tweets en inglés como para tweets en español. En este capítulo se describe la metodología que se ha utilizado para realizar esta tarea, detallando todos los pasos que se han seguido para llevarla a cabo.

3.1 Procedimiento seguido

Los pasos a seguir para llevar a cabo la clasificación de tweets son: preprocesar el texto, obtener los vectores de palabras e implementar diferentes máquinas (algoritmos) que nos ayudarán con la clasificación.

- **Obtención de datos:** los tweets en inglés los hemos obtenido de un dataset de *Kaggle* [25], una comunidad en línea sobre aprendizaje automático. Para los tweets en español, hemos utilizado la web de *TASS (Taller de Análisis Semántico en la Sociedad Española para el Procesamiento del Lenguaje Natural)* [26], en la cual existen ficheros XML con tweets en español.
- **Preprocesamiento del texto:** consiste en limpiar todo aquello que no es en sí texto, es decir, enlaces, usuarios, *hashtags*, caracteres no alfanuméricos, codificaciones, palabras sin significado o *stopwords* (*donde*, *de*, *a*, *ni*, *nada*, etc) … Se trata de normalizar el texto.
- **Vectores de palabras:** las máquinas no reciben palabras para realizar clasificaciones, sino vectores. De alguna forma hay que realizar una representación numérica de las palabras para que los algoritmos de clasificación puedan entenderlas.
- **Implementación de clasificadores:** una vez obtenidos los vectores con los cuales representamos cada una de las palabras, pasamos éstos a los algoritmos de Machine Learning para que podamos entrenarlos y obtengamos modelos con los cuales podamos realizar nuevas clasificaciones de textos. Habrá que realizar diferentes ensayos con estos modelos para así poder optimizarlos.
- **Evaluación de resultados:** por último, tras haber realizados los diferentes experimentos, se muestran los resultados obtenidos en cada uno de ellos y se realiza la elección de los mejores modelos de acuerdo con dichos resultados.

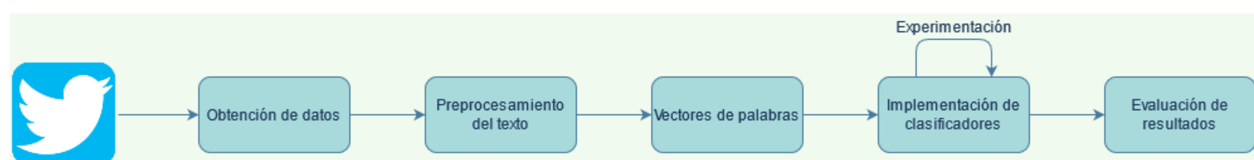

Figura 3-1. Esquema de la metodología utilizada.

3.2 Obtención de datos

El primer paso del procedimiento consiste en obtener los datos con los que vamos a partir para llevar a cabo el análisis de sentimientos. En nuestro caso hemos decidido que estos datos van a ser tweets de la red social *Twitter* [1], en la cual la gente expresa sus sentimientos y opiniones sobre cualquier tema. Por lo tanto, son muy adecuados para llevar a cabo esta tarea [27].

Para el idioma inglés, se ha partido de una base de datos existente en *Kaggle* [25], la cual contiene 1.600.000 tweets, de los cuales 800.000 están etiquetados como positivos y los otros 800.000 como negativos. Se trata de un archivo CSV que contiene en cada línea la información de un tweet.

En el caso del idioma español, no existen muchas bases de datos de tweets. Existe una web del *Taller de Análisis Semántico en la Sociedad Española para el Procesamiento del Lenguaje Natural (TASS)* [26], una asociación científica que pone a disposición archivos XML con tweets en español, a partir de la cual podemos obtener una base de datos con los mismos.

3.2.1 Tweets en inglés

Para empezar, partiremos del fichero CSV con 1.600.000 tweets en inglés. Estos tweets serán nuestro "corpus", es decir, nuestra recopilación de texto con la que partimos para realizar el análisis de sentimientos.

Este CSV lo podemos guardar en una variable en Python. Esta variable, por comodidad, será de tipo *dataframe*, la cual nos permite visualizar muy bien los datos en forma de tabla. Utilizaremos el método *read_csv()* del *dataframe* para pasar todos los datos del fichero CSV a esta variable:

```
# Lee  el fichero csv sin la cabecera y lo guarda como dataframe
df = pd.read_csv("training.1600000.processed.noemoticon.csv", encoding='latin-1',
header=None)
```

Código 3-1. Método *read_csv()*.

Un ejemplo del formato de tweet sería el siguiente:

```
0,"1467810369","Mon Apr 06 22:19:45 PDT
2009","NO_QUERY","_TheSpecialOne_","@switchfoot http://twitpic.com/2y1zl - Awww,
that's a bummer.  You shoulda got David Carr of Third Day to do it.
```

Figura 3-2. Muestra del formato de tweet en inglés.

Como vemos tiene estas columnas:

- **sentiment:** la polaridad del tweet (0 = negativo, 4 = positivo).
- **id:** el id del tweet (1467810369).
- **date:** la fecha del tweet (Mon Apr 06 22:19:45 PDT 2009).
- **query:** la consulta. Si no hay consulta, el valor es NO_QUERY.
- **user:** el usuario del tweet (_TheSpecialOne_).
- **text:** el texto del tweet (@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it.).

Como las columnas que nos son útiles para nuestro trabajo son “sentiment” y “text”, eliminaremos las restantes con el método *drop()* del *dataframe*.

Por otra parte, barajaremos las filas para que no aparezcan primero los tweets negativos seguidos de los positivos, sino que estén todos mezclados entre sí.

```
# Quita las columnas que no sirven
df = df.drop(["id", "date", "query", "user"], axis = 1)

# Baraja las filas del dataframe
np.random.seed(0)
df = df.iloc[np.random.permutation(df.index)].reset_index(drop=True)
```

Código 3-2. Método *drop()* y barajamiento de tweets.

Una muestra de los diez primeros tweets barajados sería la siguiente:

	sentiment	text
0	0	wants to compete! i want hard competition! i want to rally. i want to feel the power coming out of the engine! i want to compete
1	0	It seems we are stuck on the ground in Amarillo. They have put a ground stop for all flights leaving for Denver. Said updates in an hour
2	0	where the f are my pinking shears? rararararrrararr...babyproofing while cutting stuff makes me stick shears random places & forget them
3	0	0ff t0 tHE MEEtiN.. i HAtE WhEN PPl V0lUNtEER MY fREE tiME..gRRR!
4	4	@ reply me pls
5	0	@bharathy_99: Jazz in India is just Honda strategy to prove they can make affordable cars for the working class. It still doesn't fit me.
6	4	aaaaaaaaaaah, met a boy. he seems nice. im happppppy now
7	4	@jonasbrothers http://twitpic.com/6q1om - Sports Center !!!!! you guys are too legit to quit. Wooww
8	4	@saragarth Not bad, bit grumpy cause of exams but generally OK ta x
9	0	@luke_redroot can't watch it what is it?

Figura 3-3. Muestra de tweets en inglés barajados.

Si realizamos el diagrama de caja de los tweets, obtenemos que la mayoría de ellos tienen entre 40 y 100 caracteres, con una longitud media de 70 caracteres, tal y como podemos ver en la Figura 3-4. Algunos de ellos superan los 280 caracteres, que es lo máximo que permite *Twitter*, pero esto es debido a que muchos caracteres están codificados con su correspondiente código y, de esta forma, se produce un aumento en la longitud del tweet.

```
# Anade una nueva columna "pre_clean_len" al dataframe con la longitud de cada tweet
df['pre_clean_len'] = [len(t) for t in df.text]

# Muestra el diagrama de caja de la columna "pre_clean_len"
plt.boxplot(df.pre_clean_len)
plt.show()
```

Código 3-3. Representación del diagrama de caja de la longitud de los tweets.

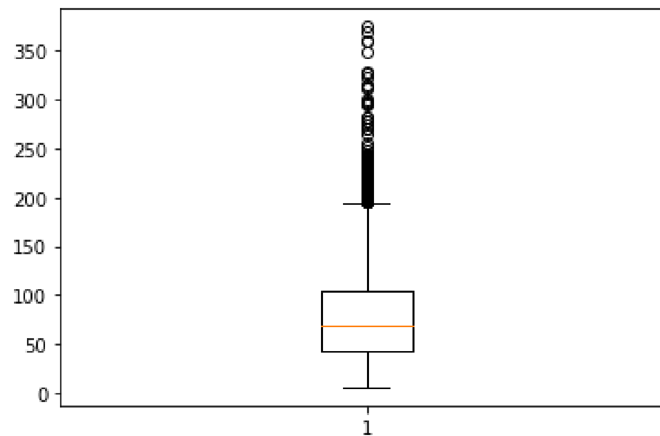

Figura 3-4. Diagrama de caja de tweets en inglés.

3.2.2 Tweets en español

En este caso utilizaremos la web de *TASS*, en la cual se nos proporcionan archivos XML con tweets en español que se han escrito en los últimos años.

Una vez unidos estos ficheros en uno solo, utilizaremos un código en Python que pasa de fichero XML a fichero CSV. Hay multitud de códigos en Internet que realizan esto, hemos seleccionado uno de ellos para llevar a cabo esta tarea [28].

Obtenemos finalmente un fichero CSV con 13.440 tweets en español (aproximadamente la mitad con polaridad positiva y la otra mitad con negativa)

```
"142926609215799296","anabelenroy_tve","Tranquilidad en las carreteras en el segundo
día de Puente de la Constitución - http://t.co/NRk2uftQ","2011-12-03T12:22:13","P"
```

Figura 3-5. Muestra del formato de tweet en español.

En este caso, al leer del fichero los tweets para pasarlos a un *dataframe*, las columnas que obtenemos son:

- **tweetid:** el id del tweet (142926609215799296).
- **user:** el usuario del tweet (anabelenroy_tve).
- **text:** el texto del tweet (Tranquilidad en las carreteras en el segundo día de Puente de la Constitución - http://t.co/NRk2uftQ).
- **date:** la fecha del tweet (2011-12-03T12:22:13).
- **sentiment:** la polaridad del tweet (P = positivo, N = negativo, NONE = nulo, NEU = neutral).

Al igual que hicimos con los tweets en inglés, eliminaremos del *dataframe* las columnas que no nos sirven y nos quedaremos con las que contienen el texto y la polaridad del sentimiento.

Como en este caso la columna "sentiment" puede tener cuatro tipos de polaridad, eliminaremos todas aquellas filas que tengan polaridad nula (NONE) y neutra (NEU). Además, cambiaremos la P por un 1 y la N por un 0, para simplificar las operaciones futuras.

De la misma forma que hicimos anteriormente, mezclamos entre sí las filas y obtenemos los siguientes diez primeros tweets:

	text	sentiment
0	#QueFeoTuConsejoOe no funciona tu huevada	0
1	Siempre la paz es lo mejor. De lo sucedido en Colombia ayer, duele es que solamente 36% de la gente salió a votar. Ganó la indiferencia	0
2	La Iglesia lanza su campaña para crear empleo. Son los únicos q creará Rajoy con la reforma laboral. #25M #29malacalle	0
3	@Josvach no fui al último concierto pq nadie me quería acompañar	0
4	Hay videos que no se envían porque no tengo espacio en el móvil así que lo siento	0
5	Éste es el "nuevo" y fantasmagórico hospital de Toledo... Varios años después (vea las fotos) - http://t.co/RPgYvq99	0
6	Juro que todo lo veo borroso	0
7	Buenos días!!! :-)hoy regreso a casa!	1
8	Pasar por mi barrio y ver a #rosadiez en la carteleria pero rota o partida en dos, me hace pensar que será así como esta #upyd	0
9	Brother en verdad me duele el cuerpo y no tengo la más mínima idea del porqué	0

Figura 3-6. Muestra de tweets en español barajados.

Si realizamos el diagrama de caja como hicimos anteriormente en el idioma inglés, obtenemos que los tweets en español están comprendidos mayoritariamente entre 70 y 130 caracteres y tienen una longitud media de 110 caracteres:

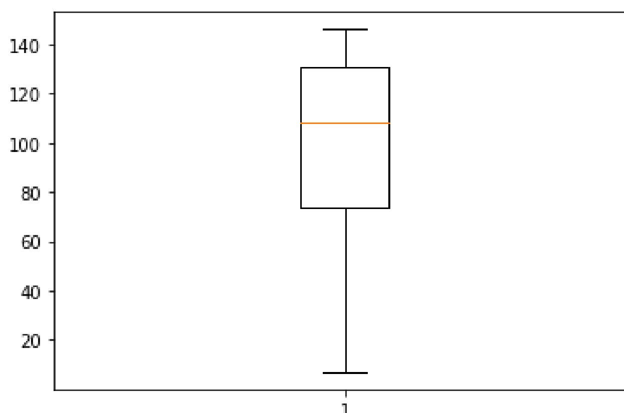

Figura 3-7. Diagrama de caja de tweets en español.

3.3 Preprocesamiento del texto

Una vez obtenidos los tweets con los que vamos a partir, pasamos a realizar una normalización del texto, realizando para ello una limpieza del mismo.

3.3.1 Tweets en inglés

Si observamos la Figura 3-3, a simple vista podemos ver que los tweets pueden estar escritos tanto en mayúsculas como en minúsculas, con signos de puntuación y caracteres no alfanuméricos, incluyendo tanto enlaces como codificación de texto. De igual forma, hay palabras que no nos aportan significado para diferenciar unos tweets de otros (*to*, *i*, *the*, *for*, *a*…). Es por ello por lo que hay que realizar un preprocesamiento de los tweets, para que de esta manera todos estén normalizados de la misma forma (pasándolos a minúsculas, eliminando *stopwords* (palabras sin significado), signos de puntuación, codificaciones…) [29].

Para limpiar el texto, utilizaremos el módulo de Python *re* [16]. Este módulo nos permite eliminar texto que coincida con un patrón o sustituirlo por otro cualquiera.

Un ejemplo de patrones que podemos utilizar son los siguientes:

```
# Elimina las menciones con @ de los tweets
pat1 = r'@[A-Za-z0-9_]+'

# Elimina los hashtags de los tweets
pat2 = r'#[A-Za-z0-9]+'

# Elimina las URL's que comienzan por "http" o https" de los tweets
pat3 = r'https?://[^ ]+'
```

Código 3-4. Ejemplo de patrones de la librería *re* (I).

El primero de ellos busca cualquier palabra que comience por el carácter "@" seguido de una o varias letras (+), números o caracteres guion bajo "_". De esta forma conseguimos eliminar los nombres de usuarios de los tweets. El segundo actúa de igual forma que el primero, pero en vez de buscar el carácter "@" busca "#" para eliminar los *hashtags*. El tercero busca una palabra que comience por http o https (el carácter "?" indica que el anterior puede o no aparecer en la búsqueda), seguido de dos puntos (:), dos barras (//) y cualquier carácter o conjunto de caracteres (+) que no comiencen por un espacio en blanco ([^]) para así eliminar las URLs.

Utilizando la función *re.sub()* de esta librería, podemos pasarle estos patrones que hemos definido anteriormente junto con el texto a limpiar y sustituirlos si los encuentra por el carácter nulo, por ejemplo (‘’). De esta forma conseguimos eliminar usuarios, *hashtags* y URLs.

Un ejemplo de uso de esta función es la siguiente:

```
# Sustituye los caracteres que no sean letras por espacios en blanco
letters_only = re.sub("[^a-zA-Z]", " ", join_compound)
```

Código 3-5. Ejemplo de patrones de la librería *re* (II).

En este caso estamos sustituyendo en la cadena "join_compound" todos los caracteres que no sean letras ([^a-zA-Z]) por espacios en blanco (" "). O, incluso, podemos crear patrones más complejos como el siguiente:

```
# Elimina tres o mas caracteres repetidos
repeated_letter = re.sub("(.)\\1{2,}", "\\1\\1", letters_only)
```

Código 3-6. Ejemplo de patrones de la librería *re* (III).

En este caso estamos buscando un carácter, el cual lo etiquetamos con un 1, que se repita tres o más veces ({2,}). Si lo encuentra, lo sustituye por éste mismo repetido dos veces (\\1\\1). Esto se realiza porque en el idioma inglés no existen palabras en las que aparezca más de dos veces un carácter repetido. Por este motivo, si en algún tweet aparece escrita una palabra como "goooood", la cambiaría por "good".

De igual forma, podemos crear otros patrones para dejar completamente el texto limpio o incluso para sustituir contracciones como "isn't" por "is not". Esto último es útil realizarlo porque podemos encontrarnos tweets con contracciones y otros en los que la misma contracción está expandida. Nuestra máquina consideraría que estas dos palabras son diferentes, aunque realmente son las mismas. Es por ello por lo que hay que normalizarlas, utilizando un diccionario de contracciones en el que aparezcan cada una de ellas con sus respectivas expansiones.

Por otro lado, también podemos eliminar codificación de texto como "&" o """ utilizando en este caso la librería *BeautifulSoup*. Con sus métodos *get_text()* y *decode()* podemos eliminar codificación tanto en

HTML como en UTF-8.

Por último, también es importante poner todo el texto en minúsculas, para que la máquina entienda que palabras como “Hello”, “hello” y “hElLO” son la misma. Esto podemos realizarlo con el método *lower()* de las cadenas de Python.

En nuestro caso hemos creado la siguiente función que recibe los tweets y realiza estas modificaciones que se han ido comentando anteriormente:

```
# Define la funcion "re_cleaner" para limpiar los tweets
def re_cleaner(text):

    # Crea el objeto de tipo "BeautifulSoup"
    soup = BeautifulSoup(text, 'lxml')

    # Extrae solo el texto de los tweets
    souped = soup.get_text()

    # Elimina la codificacion utf-8-sig
    try:
        bom_removed = souped.decode("utf-8-sig").replace(u"\ufffd", "?")
    except:
        bom_removed = souped

    # Elimina el combined_pat
    stripped = re.sub(combined_pat, '', bom_removed)

    #Elimina las URL's
    stripped = re.sub(www_pat, '', stripped)

    # Convierte todo en minusculas
    lower_case = stripped.lower()

    # Expande las contracciones
    cont_handled = cont_pattern.sub(lambda x: contraction_map[x.group()],
lower_case)

    # Elimina el guion entre palabras compuestas y las une entre si
    join_compound = re.sub("([A-Za-z]+)-([A-Za-z]+)", "\\1\\2", cont_handled)

    # Sustituye los caracteres que no sean letras por espacios en blanco
    letters_only = re.sub("[^a-zA-Z]", " ", join_compound)

    # Elimina tres o mas caracteres repetidos
    repeated_letter = re.sub("(.)\\1{2,}", "\\1\\1", letters_only)

    # Tokeniza las palabras y solo considera aquellas cuya longitud sea mayor de 1
    words = [x for x  in tok.tokenize(repeated_letter) if len(x) > 1]

    # Une las palabras y las devuelve
    return (" ".join(words)).strip()
```

Código 3-7. Función *re_cleaner()* en el caso inglés.

Como resultado obtenemos lo siguiente en los primeros tweets:

```
['wants to compete want hard competition want to rally want to feel the power coming out of the engine want to compete',
 'it seems we are stuck on the ground in amarillo they have put ground stop for all flights leaving for denver said updat
es in an hour',
 'where the are my pinking shears rarararrarararr babyproofing while cutting stuff makes me stick shears random places fo
rget them',
 'ff the meetin hate when ppl lunteer my free time grr',
 'reply me pls',
 'jazz in india is just honda strategy to prove they can make affordable cars for the working class it still does not fit
me',
 'aah met boy he seems nice im happy now',
 'sports center you guys are too legit to quit wooww',
 'not bad bit grumpy cause of exams but generally ok ta',
 'cannot watch it what is it',
 'good nite everbody had long day and did project now it is finally time to get some rest peace',
 'morning twitter world am gonna start my day with the coldest lucozade can find',
 'hahaa thats really random but hi',
 'because they are just awesome if you wanna record one thenn go to recording studio haha bruce records people',
 'forcei participar do twitter',
 'kutnerr why why and to think that is still on the show ugh kutner kal penn you have been the bright star in ho',
```

Figura 3-8. Muestra de tweets en inglés (Figura 3-3) tras haber sido preprocesados con la librería *re*.

Una vez limpio todo el texto con la librería *re*, pasamos a *tokenizarlo*. *Tokenizar* significa separar las palabras unas de otras para tratarlas como unidades, sin tener en cuenta las adyacentes. Esto lo realizaremos para corregir la ortografía posteriormente, eliminar *stopwords* y realizar *stemming*.

Las *stopwords* son palabras que no tienen un significado por sí solas, sino que modifican o acompañan a otras. Es decir, no nos proporcionan información que nos permita distinguir tweets con polaridad positiva de tweets con polaridad negativa. Este grupo suele estar conformado por artículos, pronombres, preposiciones, adverbios e incluso algunos verbos.

Stemming es un método que consiste en cambiar las palabras por su raíz. De esta forma, si tenemos palabras como “wants” o “wanted”, las cambiaríamos por “want” y así a la máquina le pasaremos una palabra en vez de dos distintas, ya que ambas significan lo mismo, aunque estén en diferentes tiempos verbales. Nos quedaremos entonces con las raíces de las palabras, las cuales recogen el significado global de las mismas.

Comenzaremos *tokenizando* las palabras, utilizando para ello la función *word_tokenize()* que nos proporciona la librería *NLTK* [15] y las iremos metiendo en una lista.

```
from nltk.tokenize import word_tokenize
nltk.download('punkt')

# Inicializa la lista de tokens
word_tokens = []

# Tokeniza cada palabra de la lista "re_clean_tweets" y las anade a la nueva
for word in re_clean_tweets:
    word_tokens.append(word_tokenize(word))
```

Código 3-8. *Tokenización* de palabras.

Lo que obtenemos es una lista con todas las palabras *tokenizadas*, separadas unas de otras dentro de los tweets:

```
[['wants','to','compete','want','hard','competition','want','to','rally','want','to','feel','the','power','coming','out',
  'of','the','engine','want','to','compete'],['it','seems','we','are','stuck','on','the','ground','in','amarillo','they',
  'have','put','ground','stop','for','all','flights','leaving','for','denver','said','updates','in','an','hour'],['where',
  'the','are','my','pinking','shears','rararararararr','babyproofing','while','cutting','stuff','makes','me','stick',
  'shears','random','places','forget','them'],['ff','the','meetin','hate','when','ppl','lunteer','my','free','time','grr'],
 ['reply','me','pls'],...
```

Figura 3-9. Muestra de palabras *tokenizadas* de los tweets en inglés (Figura 3-8).

Cuando ya tengamos la lista de palabras *tokenizadas*, utilizaremos la librería *SymSpell* [18] para corregirlas ortográficamente. En los tweets, podemos encontrarnos con faltas de ortografía o errores en las palabras o, incluso, palabras como “amaziiiing” que pasaría a ser “amaziing” tras haberla limpiado con la función definida en el código 3-7, sigue siendo incorrecta. Por ello vamos a utilizar un corrector ortográfico. Hemos seleccionado *SymSpell* puesto que es más rápido que otros existentes y las correcciones son buenas.

```
import pkg_resources
from symspellpy import SymSpell, Verbosity

# Busca el diccionario y lo configura
sym_spell = SymSpell(max_dictionary_edit_distance=2, prefix_length=7)
dictionary_path = pkg_resources.resource_filename("symspellpy",
"frequency_dictionary_en_82_765.txt")

# "term_index" es la columna del termino y "count_index" es la de la frecuencia
sym_spell.load_dictionary(dictionary_path, term_index=0, count_index=1)

# Busca la correccion de la palabra
# Si es correcta o no encuentra correccion devuelve la palabra original
correct_tokens = []
for m in word_tokens:
    correct_word = [sym_spell.lookup(w, Verbosity.TOP, include_unknown=True)[0].term
for w in m]
    correct_tokens.append(correct_word)
```

Código 3-9. Corrección de palabras.

Tras realizar las correcciones de palabras, pasamos a eliminar las *stopwords*. La librería *NLTK* nos proporciona un diccionario con estas palabras. Recorremos todos los tweets buscando si existen *stopwords* en ellos y las eliminaremos de nuestra lista de palabras *tokenizadas* y corregidas.

```
from nltk.corpus import stopwords

# Decarga las stopwords del idioma ingles
nltk.download('stopwords')
stop = set(stopwords.words('english'))

# Inicializa la lista
stop_tokens = []

# Solo guarda en la lista las palabras que no son stopwords
for m in correct_tokens:
    a = [w for w in m if not w in stop]
    stop_tokens.append(a)
```

Código 3-10. Eliminación de *stopwords* en inglés.

Por último, podemos realizar *stemming* a las palabras. Para realizar *stemming* utilizaremos la clase *PorterStemmer* de la librería *NLTK* y actualizaremos nuestra lista de palabras con las raíces de éstas.

```
from nltk.stem import PorterStemmer

if STEM == True:
    # Inicializa la lista
    stem_tokens = []

    # Crea el objeto "PorterStemmer"
    ps = PorterStemmer()

    # Guarda en la lista las raices de las palabras
    for l in stop_tokens:
        b = [ps.stem(q) for q in l]
        stem_tokens.append(b)
```

Código 3-11. Realización de *stemming* en inglés.

Una vez realizados todos estos pasos y construidas de nuevo las frases con sus respectivas palabras, nuestros tweets quedan totalmente limpios tal y como se muestra en la Figura 3-10:

```
['want compet want hard competit want ralli want feel power come engin want compet',
 'seem stuck ground amarillo put ground stop flight leav denver said updat hour',
 'pink shear rarararrararrr babyproof cut stuff make stick shear random place forget',
 'meet hate pal hunter free time err',
 'repli plu',
 'jazz india honda strategi prove make afford car work class still fit',
 'aah met boy seem nice happi',
 'sport center guy legit quit wood',
 'bad bit grumpi caus exam gener',
 'watch',
 'good site everybodi long day project final time get rest peac',
 'morn twitter world start day coldest lucozad find',
 'sahara realli random',
 'awesom wan record one go record studio hama bruce record peopl',
 'forc particular twitter',
 'kutnerr think still show ugh hunter al penn bright star',
 'awe love new song last saw funni thursday live chat job serious',
 'tylenol work best',
 'song life love liesimpl plan beautifulylost',
```

Figura 3-10. Muestra de tweets en inglés (Figura 3-8) tras realizar corrección ortográfica, eliminación de *stopwords* y *stemming*.

3.3.1.1 Nubes de palabras

Por último, también es útil añadir dos *WordClouds* con la librería *wordcloud* [20] de Python para así de esa forma poder ver de forma rápida cuáles son las palabras más repetidas tanto en los tweets con polaridad negativa (Figura 3-11 izquierda) como en los de polaridad positiva (Figura 3-11 derecha).

```
from wordcloud import WordCloud

# Genera wordcloud de sentimientos negativos
wordcloud = WordCloud(width=3000, height=2000, colormap='Set2').generate('
'.join(clean_df['text'][clean_df["target"] == 0]))

# Dibuja la imagen WordCloud
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

Código 3-12. Representación de *WordCloud*.

Figura 3-11. Nubes de palabras de tweets en inglés con polaridad negativa (izquierda) y polaridad positiva (derecha).

Podemos apreciar que muchas de ellas se repiten tanto en una clase como en la otra. ¿A qué es debido esto? Quizás se deba a la ironía o sarcasmo de los tweets. Una palabra como puede ser “love” (“amor”) la asociamos instintivamente a algo bueno. Le otorgamos una connotación positiva. Pero cuando un usuario escribe un tweet con la palabra “love” puede mostrar tanto un sentimiento positivo (“I love the gift that my friends have given me”) (“Me encanta el regalo que me han hecho mis amigos”) como uno negativo, utilizando para ello la ironía (“I love how well the seller has wrapped my package”) (“Me encanta lo bien que el vendedor ha envuelto mi paquete”). Aquí está ese problema que existe en hacer que las máquinas puedan “pensar” por sí mismas. Lo difícil de que una máquina pueda distinguir si algo causa sentimiento negativo o positivo reside en la capacidad de entender la ironía o el sarcasmo en las oraciones. Otro ejemplo es la expresión “Don’t be so angry! She was only pulling your leg!” (“¡No te enfades! ¡Te está tomando sólo el pelo!”). La palabra “angry” (“furioso”) tiene connotación negativa y por lo tanto un sentimiento negativo. Sin embargo, en este contexto está animando a una persona a que esté alegre y que es una broma. Se debe interpretar como un sentimiento positivo, de ánimo [30].

Por otro lado, hay palabras que por sí solas no podemos asociarlas a algo bueno o malo. Por ejemplo, otra palabra que se repite en ambas clases es “today” (“hoy”). Por sí sola no podemos clasificarla como sentimiento positivo o negativo, tenemos que asociarla a un contexto. Podríamos decir “Today is a good day” (“Hoy es un buen día”). En este caso podemos decir que el contexto es positivo y por lo tanto la clasificaremos como tal. Pero… ¿qué pasaría si utilizamos “today” como algo negativo? “Today I lost my wallet” (“Hoy perdí mi cartera”). Esta frase causa sentimiento negativo. Hay, por lo tanto, palabras que pueden ser usadas tanto de forma positiva como negativa, dependen del contexto en el que se encuentren. Por consiguiente, pueden aparecer tanto en tweets positivos como en negativos, indistintamente.

3.3.2 Tweets en español

Una vez realizado el preprocesamiento de tweets en inglés vamos a llevar a cabo lo mismo, pero con los tweets en español.

Comenzamos preprocesando el texto con la librería *re*. Eliminamos usuarios, *hashtags*, URLs, codificación HTML y UTF-8, pasamos a minúsculas… Todo igual que hicimos en inglés. Lo que hacemos diferente en este caso es que suprimimos toda la parte de la expansión de contracciones en inglés y añadimos la eliminación de tildes. Esto último es muy sencillo de realizar puesto que el método *unidecode()* de la librería *unidecode* [19] nos ayuda con esta tarea.

```
# Quita las tildes
rem_tilde = unidecode.unidecode(lower_case)
```

Código 3-13. Eliminación de tildes.

Los tweets quedarían tal que así:

```
['no funciona tu huevada',
 'siempre la paz es lo mejor de lo sucedido en colombia ayer duele es que solamente de la gente salio votar gano la indif
erencia',
 'la iglesia lanza su campana para crear empleo son los unicos creara rajoy con la reforma laboral',
 'no fui al ultimo concierto pq nadie me queria acompanar',
 'hay videos que no se envian porque no tengo espacio en el movil asi que lo siento',
 'este es el nuevo fantasmagorico hospital de toledo varios anos despues vea las fotos',
 'juro que todo lo veo boroso',
 'buenos dias hoy regreso casa',
 'pasar por mi bario ver en la carteleria pero rota partida en dos me hace pensar que sera asi como esta',
 'brother en verdad me duele el cuerpo no tengo la mas minima idea del porque',
 'pasando un tarde hermosa en mercedes mirando la ciudad desde lo mas alto',
 'jajajaja la tuya la mucha gente seguro pero yo no puedo sin mi melena me muero',
 'no para esa hora es devastador el efecto de en mi pais en realidad',
 'enhorabuena al bilbao grande',
 'lo bueno de la vida es que te hace coincidir con personas realmente increibles',
 'manana volvera estar con los ciudadanos de para invitarles tomar la palabra gta ejercito metro carabanchel am',
 'estos dias tratare de hacer deporte con mis amigos desconectar',
 'el quiere legar al poder sin decir nada dice en califica esta actuacion de estafa democratica',
```

Figura 3-12. Muestra de tweets en español (Figura 3-6) tras haber sido preprocesados con la librería *re*.

A continuación, *tokenizamos* las palabras, eliminamos las *stopwords* (*NLTK* además de proporcionarnos el diccionario en inglés también nos lo proporciona en español) y realizamos *stemming*. Para realizar *stemming* en español utilizamos la clase *SnowballStemmer* de la librería *NLTK*.

```
from nltk.corpus import stopwords

# Decarga las stopwords del idioma espanol
nltk.download('stopwords')
stop = set(stopwords.words('spanish'))

# Inicializa la lista
stop_tokens = []

# Solo guarda en la lista las palabras que no son stopwords
for m in word_tokens:
    a = [w for w in m if not w in stop]
    stop_tokens.append(a)
```

Código 3-14. Eliminación de *stopwords* en español.

```
from nltk.stem import SnowballStemmer

if STEM == True:
    # Inicializa la lista
    stem_tokens = []

    # Crea el objeto "SnowballStemmer"
    stemmer = SnowballStemmer('spanish')

    # Guarda en la lista las raices de las palabras
    for l in stop_tokens:
        b = [stemmer.stem(q) for q in l]
        stem_tokens.append(b)
```

Código 3-15. Realización de *stemming* en español.

Una vez unidas las palabras en las frases, obtenemos lo siguiente:

```
['funcion huev',
 'siempr paz mejor suced colombi ayer duel sol gent sali vot gan indiferent',
 'iglesi lanz campan cre emple unic cre rajoy reform laboral',
 'ultim conciert pq nadi queri acompan',
 'vide envi espaci movil asi sient',
 'nuev fantasmagor hospital toled vari anos despu vea fot',
 'jur veo boros',
 'buen dias hoy regres cas',
 'pas bari ver carteleri rot part dos hac pens ser asi',
 'broth verd duel cuerp mas minim ide',
 'pas tard hermos merced mir ciud mas alto',
 'jajajaj much gent segur pued melen muer',
 'hor devast efect pais realid',
 'enhorabuen bilba grand',
 'buen vid hac coincid person realment increibl',
 'manan volver ciudadan invit tom palabr gta ejercit metr carabanchel am',
 'dias tratar hac deport amig desconect',
 'quier leg pod dec dic calif actuacion estaf democrat',
 'vam juev',
```

Figura 3-13. Muestra de tweets en español (Figura 3-12) tras realizar eliminación de *stopwords* y *stemming*.

3.3.2.1 Nubes de palabras

Por último, añadimos también los dos *Wordclouds* para ver las palabras más repetidas en ambas clases (izquierda negativo, derecha positivo):

Figura 3-14. Nubes de palabras de tweets en español con polaridad negativa (izquierda) y polaridad positiva (derecha).

3.4 Vectores de palabras

Una vez ya hemos limpiado los tweets, pasamos a la vectorización de palabras. La vectorización de palabras consiste en representar de forma numérica las mismas. Cada una de ellas se representará con un vector de números para más tarde pasar éstos a nuestra máquina, la cual realizará el aprendizaje de sentimientos con polaridad positiva y negativa.

Para realizar estos vectores de palabras, utilizaremos el paquete *Word2Vec* de la librería *Gensim* [31]. Tal y como se vio en el capítulo 2, el propósito de *Word2Vec* es que dos palabras que comparten contextos similares también comparten un significado similar y, en consecuencia, una representación vectorial similar del modelo. Por ejemplo: "coche", "camión" y "autobús" se utilizan a menudo en situaciones similares, con palabras circundantes similares como "transporte" o "vehículo", y según *Word2Vec* compartirán por tanto una representación vectorial similar.

A partir de este supuesto, *Word2Vec* puede utilizarse para averiguar las relaciones entre las palabras de un conjunto de datos, calcular la similitud entre ellas o utilizar la representación vectorial de esas palabras como entrada para otras aplicaciones, como la clasificación de textos o la agrupación.

El marco *Gensim*, creado por Radim Řehůřek, consiste en una implementación robusta, eficiente y escalable del modelo *Word2Vec*. Primero *tokenizaremos* nuestro corpus normalizado y luego construiremos el modelo *Word2Vec*. La idea básica es proporcionar un corpus de documentos como entrada y obtener vectores de características a la salida.

Internamente, construye un vocabulario basado en los documentos de entrada y aprende representaciones vectoriales de las palabras, como también vimos en el capítulo 2. Una vez completado esto, construye un modelo que puede utilizarse para extraer vectores de palabras para cada palabra de un documento.

Los parámetros principales a configurar del modelo son:

- **size (int):** establece el tamaño o la dimensión de los vectores de palabras y puede oscilar entre decenas y miles. Por defecto tiene un valor de 100.
- **window (int):** establece el contexto o el tamaño de la ventana que especifica la longitud de la ventana de palabras que debe considerarse para que el algoritmo las tenga en cuenta como contexto al entrenar. Es decir, establece cuántas palabras por delante y por detrás de la actual se incluirán como contexto de la misma. Por defecto tiene un valor de 5.
- **min_count (int):** especifica la frecuencia mínima de una palabra en todo el corpus para que sea considerada en el vocabulario. Esto ayuda a eliminar palabras muy específicas que pueden no tener mucha importancia ya que aparecen muy raramente en los documentos. Por defecto tiene un valor de 5.
- **epochs (int):** número de iteraciones sobre el corpus. Por defecto tiene un valor de 5.

```
# Crea el modelo "Word2Vec" con las palabras tokenizadas
w2v_model = Word2Vec(tokenized_tweet, size=200)

w2v_model.train(tokenized_tweet, total_examples=len(clean_df['text']),
epochs=w2v_model.epochs)
```

Código 3-16. Creación y entrenamiento del modelo *Word2Vec*.

3.4.1 Experimentación de ejemplo

Realizaremos unos experimentos como ejemplo para mostrar cómo *Word2Vec* es capaz de "captar" el significado de las palabras [32]. Para hacer estos experimentos, utilizaremos el dataset en inglés, puesto que tenemos muchos más tweets en este idioma que en español, para así obtener mejores resultados. Vamos a dejar todos los parámetros a sus valores por defecto menos "size", que pasará a ser 300, para aumentar las dimensiones de los vectores.

3.4.1.1 Palabras similares

Vamos a pedirle al modelo que nos diga cuáles son las palabras más similares a "car" (coche):

```
w2v_model.wv.most_similar(positive=["car"])

[('truck', 0.744271993637085),
 ('motorcycle', 0.6722447872161865),
 ('van', 0.5979344248771667),
 ('bike', 0.5977499485015869),
 ('brakes', 0.5946865081787109),
 ('jeep', 0.5892350673675537),
 ('garage', 0.5846748948097229),
 ('tire', 0.581353425979614З),
 ('gas', 0.5515974164009094),
 ('driver', 0.5482298135757446)]
```

Figura 3-15. Palabras más similares a "car".

Podemos observar que el modelo ha realizado un aprendizaje con todo el corpus que le hemos pasado y nos saca qué palabras tienen relación con "car". Palabras como "truck" (camión), "motorcycle" (moto), "van" (furgoneta), "bike" (bicicleta)… aparecen en esta lista, lo cual nos indica que el modelo ha sido capaz de "captar" el significado de la palabra "car".

Podemos realizar otro ejemplo con la palabra "cake" (pastel):

```
w2v_model.wv.most_similar(positive=["cake"])
```

```
[('cupcakes', 0.7253482341766357),
 ('cakes', 0.697631239891052),
 ('brownies', 0.6836669445037842),
 ('cookies', 0.6803901791572571),
 ('cheesecake', 0.6579111127853394),
 ('muffins', 0.649151086807251),
 ('pudding', 0.6453644633293152),
 ('carrot', 0.627605676651001),
 ('biscuits', 0.627117395401001),
 ('frosting', 0.6083176732063293)]
```

Figura 3-16. Palabras más similares a “cake”.

Al igual que antes, nos aparecen palabras relacionadas con “cake”: “cupcakes”, “brownies”, “cookies”, “cheesecake” (tarta de queso)… Es curioso ver cómo el modelo es capaz de mostrarnos palabras que estén relacionadas con los pasteles.

Un último ejemplo de este método con la palabra “amazing” (increíble):

```
w2v_model.wv.most_similar(positive=["amazing"])
```

```
[('awesome', 0.7913191318511963),
 ('incredible', 0.7710421681404114),
 ('fantastic', 0.7109206914901733),
 ('great', 0.6224313974380493),
 ('stunning', 0.5913349390029907),
 ('fab', 0.5904736518859863),
 ('brilliant', 0.5849590301513672),
 ('phenomenal', 0.5307651162147522),
 ('rocked', 0.5288257598876953),
 ('fabulous', 0.5195225477218628)]
```

Figura 3-17. Palabras más similares a “amazing”.

Todos estos valores que aparecen al lado de cada palabra se llaman “similitud coseno”. Cuanto más cerca de 1, más parecido hay entre las palabras (el ángulo formado entre los vectores correspondientes a cada una de ellas es más pequeño).

3.4.1.2 Similitudes entre palabras

Podemos ver, también, la similitud que hay entre dos palabras:

```
w2v_model.wv.similarity("computer", "laptop")
```

```
0.82014304
```

Figura 3-18. Similitud entre las palabras “computer” y “laptop”.

"Computer" (ordenador) y "laptop" (portátil) son dos palabras que tienen mucha relación, por lo cual están muy cercanas entre sí en el modelo. Sin embargo, probemos las dos siguientes:

```
w2v_model.wv.similarity("computer", "phone")

0.54630065
```

Figura 3-19. Similitud entre las palabras "computer" y "phone".

"Computer" y "phone" (teléfono) tienen relación, pero menos que en el ejemplo anterior.

Por último, si probamos dos palabras que no tienen relación entre sí, obtendremos un número cercano a 0:

```
w2v_model.wv.similarity("boy", "plane")

0.052042935
```

Figura 3-20. Similitud entre las palabas "boy" y "plane".

No hay mucha relación entre "boy" (niño) y "plane" (avión) y el modelo así nos lo indica.

3.4.1.3 Palabras sobrantes

Otro experimento que podemos realizar consiste en proporcionarle al modelo tres palabras y que nos descarte una de ellas, la que menos tiene que ver con las restantes:

```
w2v_model.wv.doesnt_match(['bottle', 'glass', 'sun'])

'sun'
```

Figura 3-21. Palabra sobrante entre "bottle", "glass" y "sun".

En este ejemplo está claro que entre "bottle" (botella), "glass" (vidrio) y "sun" (sol), la palabra que no tiene relación con las restantes es "sun".

Otro ejemplo más:

```
w2v_model.wv.doesnt_match(['spoon', 'light', 'fork'])

'light'
```

Figura 3-22. Palabra sobrante entre "spoon", "light" y "fork".

Al igual que en el ejemplo anterior, entre las palabras “spoon” (cuchara), “light” (luz) y “fork” (tenedor), la que sobra es “light”.

3.4.1.4 Analogías de palabras

Pasamos a realizar otro experimento. Éste consiste en obtener analogías de palabras. Por ejemplo:

```
w2v_model.wv.most_similar(positive=["woman", "boy"], negative=["man"], topn=3)

[('girl', 0.5133512020111084),
 ('child', 0.48823797702789307),
 ('nephew', 0.45128846168518066)]
```

Figura 3-23. Analogía de palabras entre “woman”, “boy” y “man”.

¿Qué palabra es a “woman” (mujer) como “boy” (niño) es a “man” (hombre)? El modelo nos proporciona la respuesta correcta: “girl” (niña).

Otro ejemplo más:

```
w2v_model.wv.most_similar(positive=["canada", "paris"], negative=["france"], topn=3)

[('toronto', 0.6854574680328369),
 ('montreal', 0.6581289172172546),
 ('singapore', 0.6557411551475525)]
```

Figura 3-24. Analogía de palabras entre “canada”, “paris” y “france”.

¿Qué palabra es a “canada” como “paris” a “france”? De nuevo el modelo nos proporciona la palabra: “toronto”. No es la capital, pero es una ciudad muy importante de Canadá.

3.4.1.5 Visualización con t-SNE

t-SNE es un algoritmo de reducción de la dimensionalidad no lineal que intenta representar datos de alta dimensión y las relaciones subyacentes entre vectores en un espacio de menor dimensión [33]. Tiene la característica de que vectores “cercanos” en un espacio de muchas dimensiones se transforman en vectores también “cercanos” en un espacio de dos dimensiones.

Utilizando el código del libro “Text analytics with Python” de D. Sarkar [34], podemos representar nuestros vectores de 300 dimensiones en gráficos de 2 dimensiones, y ver si podemos detectar patrones interesantes.

Para ello, vamos a utilizar la implementación *t-SNE* de *Scikit-learn.*

```
# Obtiene las palabras más similares
similar_words = {search_term: [item[0]
for item in w2v_model.wv.most_similar([search_term], topn=5)]
for search_term in ['egg', 'salad', 'strawberries', 'chicken', 'hamburger', 'sushi',
"fish"]}

import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
%matplotlib inline

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

Código 3-17. Algoritmo *t-SNE*.

Lo primero que vamos a realizar consiste en obtener las cinco palabras más similares de las siguientes palabras relacionadas con comida: “egg” (huevo), “salad” (ensalada), “strawberries” (fresas), “chicken” (pollo), “hamburger” (hamburguesa), “sushi” y “fish” (pescado). Estas palabras similares se han obtenido y se han guardado en una lista al inicio del Código 3-17:

```
similar_words

{'egg': ['scrambled', 'sausage', 'eggs', 'porridge', 'onions'],
 'salad': ['beef', 'potatoes', 'shrimp', 'salmon', 'tuna'],
 'strawberries': ['berries', 'fruit', 'ores', 'oatmeal', 'pudding'],
 'chicken': ['beef', 'pork', 'steak', 'tofu', 'curry'],
 'hamburger': ['hotdogs', 'shrimp', 'hotdog', 'broccoli', 'durian'],
 'sushi': ['tacos', 'pho', 'pizza', 'dessert', 'chipotle'],
 'fish': ['crab', 'chips', 'tomatoes', 'fishes', 'lettuce']}
```

Figura 3-25. Palabras más similares a otras relacionadas con comida.

A continuación, representamos los vectores asociados a cada una de las palabras que aparecen en la lista anterior en 2 dimensiones, tal y como se realiza en el Código 3-17. Obtenemos lo siguiente:

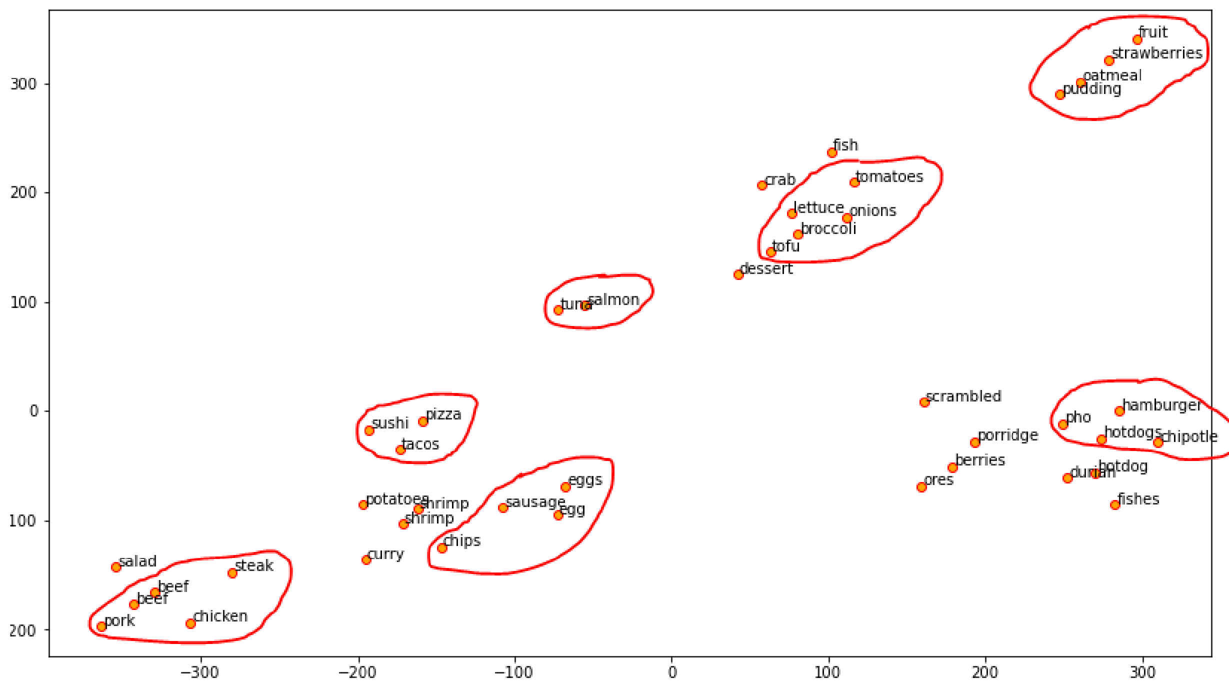

Figura 3-26. Representación en dos dimensiones de vectores de palabras con *t-SNE*.

Podemos observar que hay diferentes grupos de palabras que son interesantes, marcados con un círculo rojo. Son palabras que están representadas cerca unas de otras y que tienen relación entre sí. Gracias a la representación que obtenemos con *t-SNE* de los vectores en 2 dimensiones, podemos ver a simple vista cómo, gracias a *Word2Vec*, podemos "captar" el significado de las palabras.

3.4.2 Vectores de tweets

Hemos comprobado cómo *Word2Vec* "capta" el significado de las palabras de forma individual, pero en nuestro caso, dado que nuestros datos contienen tweets y no sólo palabras, tendremos que encontrar una manera de utilizar los vectores de palabras del modelo *Word2Vec* para crear una representación vectorial para un tweet completo. Hay una solución sencilla para este problema, podemos simplemente tomar la media de todos los vectores de palabras presentes en el tweet. La longitud del vector resultante será la misma. Repetiremos el mismo proceso para todos los tweets en nuestros datos y obtendremos sus correspondientes vectores [35].

Para simplificar las operaciones y disminuir el proceso de computación cuando pasemos a los clasificadores, el parámetro "size" de *Word2Vec*, el cual lo configuramos a 300 en el apartado anterior, lo bajaremos a 200. Esto quiere decir que nuestros vectores pasarán ahora a tener una longitud de 200.

```
# Crea el modelo "Word2Vec" con las palabras tokenizadas
w2v_model = Word2Vec(tokenized_tweet, size=200)

w2v_model.train(tokenized_tweet, total_examples=len(clean_df['text']),
epochs=w2v_model.epochs)

# Funcion para realizar la media de los vectores de palabras
def word_vector(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0
    for word in tokens:
        try:
            vec += w2v_model[word].reshape((1, size))
            count += 1.
        except KeyError:  # maneja el caso en el que el token no esté en el
vocabulario
            continue
    if count != 0:
        vec /= count
    return vec

# Para cada tweet, realiza la media de los vectores de las palabras que contiene
wordvec_arrays = np.zeros((len(tokenized_tweet), 200))
for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

Código 3-18. Creación del modelo y realización de la media de vectores de palabras con *Word2Vec*.

Otra opción que tenemos para obtener el vector de un tweet completo es utilizar *Doc2Vec*, una extension de *Word2Vec* [36], el cual describimos en el capítulo 2. La diferencia es que, en vez de obtener vectores para cada palabra, lo obtiene para cada frase.

Para implentarlo, tenemos que etiquetar cada tweet *tokenizado* con un ID único. Podemos hacerlo utilizando la función *LabeledSentence()* de *GenSim*.

```
def add_label(twt):
    output = []
    for i, s in zip(twt.index, twt):
        output.append(LabeledSentence(s, ["tweet_" + str(i)]))
    return output
```

Código 3-19. Etiquetado de tweets.

Una muestra de tweets etiquetados sería la siguiente:

```
# Muestra los primeros seis tweets etiquetados
labeled_tweets[:6]

[LabeledSentence(words=['burn', 'hand', 'hard', 'today', 'main', 'pizza', 'aug'], tags=['tweet_0']),
 LabeledSentence(words=['ton', 'topsoil', 'deal', 'today'], tags=['tweet_1']),
 LabeledSentence(words=['final', 'tire', 'morn', 'still', 'get', 'caramel', 'ice', 'coffe', 'mcdonald', 'sound', 'amaz'], t
ags=['tweet_2']),
 LabeledSentence(words=['sure', 'quadcor', 'handl', 'thing', 'well', 'go', 'emerg'], tags=['tweet_3']),
 LabeledSentence(words=['super', 'tire', 'hey', 'least', 'tuesday'], tags=['tweet_4']),
 LabeledSentence(words=['look', 'morn', 'doll', 'got', 'question'], tags=['tweet_5'])]
```

Figura 3-27. Muestra de tweets etiquetados.

Posteriormente, llamaremos a la función *Doc2Vec* para crear el modelo, pasándole el tamaño de vector (200) y obtendremos los vectores de cada uno de los tweets.

```
from gensim.models import Doc2Vec

d2v_model = Doc2Vec(vector_size=200)

d2v_model.build_vocab([i for i in tqdm(labeled_tweets)])

d2v_model.train(labeled_tweets, total_examples=len(clean_df['text']),
epochs=d2v_model.iter)

# Crea los vectores de tweets
docvec_arrays = np.zeros((len(tokenized_tweet), 200))
for i in range(len(clean_df)):
    docvec_arrays[i,:] = d2v_model.docvecs[i].reshape((1,200))

docvec_df = pd.DataFrame(docvec_arrays)
docvec_df.shape
# Etiqueta todos los tweets
labeled_tweets = add_label(tokenized_tweet)
```

Código 3-20. Creación del modelo y vectores de tweets con *Doc2Vec*.

Lo que obtenemos tanto en el caso de realizar la media de los vectores de palabras con *Word2Vec* como en la obtención directa de los vectores de tweets con *Doc2Vec* es una matriz como la siguiente:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	-0.003517	0.006306	0.027327	-0.104686	-0.127623	0.008540	-0.013914	0.013535	0.043033	0.098031	0.011887	0.026969	-0.082594	-0.069913	0.068786
1	-0.022667	0.011455	0.045685	-0.129203	-0.155034	-0.007074	-0.006763	0.011664	0.044215	0.118330	-0.013181	0.057638	-0.050994	-0.052247	0.076886
2	-0.008525	0.002936	0.032277	-0.099011	-0.096421	-0.015572	0.021132	-0.010680	0.007086	0.085617	-0.013706	0.035030	-0.060204	-0.025310	0.043776
3	-0.030119	-0.030477	0.051013	-0.072793	-0.120267	-0.087154	-0.019401	0.001102	-0.001714	0.089658	-0.098041	0.035007	-0.045571	-0.058533	0.018155
4	0.003337	0.030409	0.054543	-0.134289	-0.105109	-0.056668	-0.013097	-0.014787	-0.053164	0.079539	-0.012047	0.042178	-0.058536	-0.013972	0.029552
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
49995	0.020186	0.036385	0.077479	-0.098694	-0.073455	-0.041988	-0.038355	-0.005571	-0.077192	-0.020777	0.024818	0.070467	-0.098192	0.022438	0.002245
49996	-0.020695	-0.013951	0.027755	-0.088734	-0.096495	-0.014933	0.060747	0.017497	0.031816	0.057046	0.005291	0.037081	-0.041604	-0.027716	-0.003281
49997	-0.005726	0.011703	0.049090	-0.078614	-0.110840	0.042271	-0.020512	0.013266	0.079474	0.101098	-0.008213	-0.024651	-0.048830	-0.080151	0.036720
49998	0.032275	-0.064374	-0.018755	-0.077031	-0.076093	0.074427	0.001557	0.033954	0.039768	0.060867	0.000555	-0.010342	-0.017309	-0.059614	0.016684
49999	-0.024858	-0.008648	-0.020517	-0.133367	-0.133580	0.034283	0.024776	0.008069	0.067118	0.113723	0.039131	0.028054	-0.061064	-0.083095	0.070227

50000 rows × 200 columns

Figura 3-28. Matriz con vectores de tweets.

3.5 Implementación de clasificadores y experimentación

Una vez obtenidos los vectores de tweets, pasamos a implementar los clasificadores que utilizaremos para entrenar los tweets y encontrar los mejores modelos.

La librería *Scikit-learn* nos proporciona funciones para implementar los diferentes algoritmos de Machine Learning utilizados para entrenar los datos y obtener un modelo que clasifique otros nuevos [37].

Existen diversos algoritmos de clasificación, pero nosotros vamos a probar los dos más usados actualmente en Machine Learning: *Máquinas de Vectores de Soporte* (*Support Vector Machines, SVM*) y *Bosques Aleatorios (Random Forest)*.

Con cada uno de estos algoritmos vamos a realizar diferentes experimentos cambiando sus parámetros y

obteniendo diferentes métricas para compararlos entre sí y poder encontrar los mejores modelos.

El proceso de experimentación en Machine Learning consiste en probar los diferentes parámetros de un algoritmo (conocidos también como hiperparámetros) para así encontrar los que mejores resultados proporcionen. A la hora de experimentar con cualquiera de estos algoritmos, no existe una regla que nos permita saber qué valores de hiperparámetros funcionan mejor o peor. Un valor que funcione muy bien en un problema puede funcionar muy mal en otro. Esto es debido a que el resultado obtenido es muy dependiente del conjunto de datos que se utilice. Es por ello por lo que tenemos que probar con distintos valores de hiperparámetros para poder optimizar los resultados.

3.5.1 Experimentación previa

Tal y como se explicó en el apartado 3.2, partimos de un dataset con 1.600.000 tweets en inglés y otro con 13.440 tweets en español. El principal problema que existe para realizar un entrenamiento con un conjunto de datos tan grande como en el caso de los tweets en inglés es el tiempo de computación. Es por ello por lo que se ha decidido realizar una experimentación con menos datos para que el tiempo de computación sea aceptable. Seguramente con todo el dataset se obtengan mejores resultados, pero como este trabajo está enfocado al aprendizaje de Machine Learning y de sus conceptos, no se busca la exactitud en los mismos.

Vamos, entonces, a realizar los experimentos con un dataset de 50.000 tweets en inglés (25.000 con polaridad positiva y 25.000 con polaridad negativa) y con un dataset de 13.440 tweets en español (aproximadamente la mitad con polaridad positiva y la otra mitad con negativa).

No obstante, vamos a probar a ver si habría mucha diferencia entre utilizar 50.000 tweets y utilizar el doble, 100.000 tweets, para el caso del idioma inglés.

Para ello, compararemos la métrica "exactitud" (accuracy), la cual definiremos posteriormente, en ambos datasets. Utilizaremos como algoritmos de estos experimentos *Support Vector Machines* y *Random Forest*, en concreto, sus correspondientes implementaciones en *Scikit-learn* llamadas *SVC* [38] y *RandomForestClassifier* [39]. El algoritmo *SVM* tiene varios hiperparámetros que se explicarán con detalle posteriormente y que en este experimento vamos a dejarlos con sus valores por defecto: *C* = 1, *kernel* = 'rbf', *gamma* = 'scale'. El algoritmo *Random Forest* tiene también diferentes hiperparámetros que dejaremos con sus valores por defecto: *n_estimators* = 100, *criterion* = 'gini', *max_features* = 'auto'. Además, se realizará una separación del conjunto de datos en un conjunto de entrenamiento (90% del total) y en un conjunto de test (10% del total).

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Guarda los vectores de tweets en la variable X
X = vec_df

# Guarda las etiquetas de los sentimientos en la variable y
y = clean_df['target']

# Divide los datos en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=0)

# Entrena los datos
svc = SVC(random_state=0)
classifier = svc.fit(X_train, y_train)

# Predice los datos de test
y_pred_svc = svc.predict(X_test)

# Imprime la metrica "exactitud"
svc_accuracy = accuracy_score(y_test, y_pred_svc)
print('Accuracy:', svc_accuracy)
```

Código 3-21. Algoritmo *SVM*.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Guarda los vectores de tweets en la variable X
X = vec_df

# Guarda las etiquetas de los sentimientos en la variable y
y = clean_df['target']

# Divide los datos en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=0)

# Entrena los datos
rf = RandomForestClassifier(random_state=0)
classifier = rf.fit(X_train, y_train)

# Predice los datos de test
y_pred_rf = rf.predict(X_test)

# Imprime la metrica “exactitud”
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', rf_accuracy)
```

Código 3-22. Algoritmo *Random Forest*.

Por otra parte, también se comentó anteriormente que existen dos formas de obtener los vectores de tweets: realizando la media de los vectores de palabras con *Word2Vec* o directamente con *Doc2Vec*. Vamos a realizar los experimentos para ambos casos.

Conjunto de datos	Clasificador			
	SVM		Random Forest	
	Word2Vec	Doc2Vec	Word2Vec	Doc2Vec
50.000 tweets	0.72	0.5972	0.7068	0.5822
100.000 tweets	0.7492	0.6733	0.7283	0.6474

Tabla 3-1. Valores de la métrica “exactitud” en experimentación previa con el doble de datos.

Es lógico que con el doble de tweets obtengamos un mayor valor de la métrica “exactitud” en todos los casos. En el caso de *Word2Vec* esta mejora es pequeña. En el caso de *Doc2Vec* es mayor. El problema es que utilizar el doble de tweets implica aumentar considerablemente el tiempo de entrenamiento y, como en el caso de *Word2Vec* es donde se obtiene una mayor exactitud y hay poca diferencia entre utilizar el dataset menor y el mayor, decidimos que vamos a realizar todos los experimentos posteriores con 50.000 tweets en el caso del idioma inglés.

Otro aspecto que debemos tener en cuenta antes de comenzar con la experimentación de los clasificadores es el utilizar un conjunto de datos “stemmatizados” o sin “stemmatizar”. Tal y como se vio en el apartado 3.3, podemos realizar *stemming* y quedarnos con las raíces de las palabras o no realizarlo y dejarlas tal cual. No sabemos a priori con qué conjunto de datos los modelos darán mejores resultados, por lo que vamos a realizar

una experimentación previa para ver los resultados. Al igual que antes, lo haremos con los clasificadores *SVM* y *Random Forest* con sus hiperparámetros con valores por defecto, tanto para *Word2Vec* como para *Doc2Vec*, en inglés y en español.

Conjunto de datos	Clasificador			
	SVM		Random Forest	
	Word2Vec	Doc2Vec	Word2Vec	Doc2Vec
Stemming	0.72	0.5972	0.7068	0.5822
No *stemming*	0.7086	0.5768	0.6986	0.582

Tabla 3-2. Valores de la métrica "exactitud" en experimentación previa con conjunto *stemmatizado* en inglés.

Conjunto de datos	Clasificador			
	SVM		Random Forest	
	Word2Vec	Doc2Vec	Word2Vec	Doc2Vec
Stemming	0.6815	0.5313	0.7723	0.5513
No *stemming*	0.6555	0.5299	0.7574	0.5454

Tabla 3-3. Valores de la métrica "exactitud" en experimentación previa con conjunto *stemmatizado* en español.

Podemos ver que en el caso inglés y en cualquiera de los algoritmos, tanto para *Word2Vec* como para *Doc2Vec*, se obtiene una mayor exactitud si realizamos *stemming* al conjunto de datos. En el caso español, observamos que ocurre exactamente lo mismo. De acuerdo a estos experimentos, realizaremos tanto para español como para inglés los experimentos posteriores con el dataset al que se le haya realizado *stemming*.

Por último, cabe destacar que, en toda la experimentación anterior, se puede ver que para cualquiera de los algoritmos utilizados y para cualquiera de los experimentos realizados, siempre se obtiene una mayor exactitud en el caso de realizar las medias de los vectores de palabras de los tweets con *Word2Vec* que realizando este proceso con *Doc2Vec*. Es por ello por lo que también se decide hacer los experimentos posteriores utilizando *Word2Vec* para crear los vectores de tweets.

3.5.2 Experimentación

En este apartado se describe la experimentación a realizar sobre los conjuntos de datos, tanto en inglés como en español. Esta experimentación, como ya se comentó en el apartado 3.5.1, se realizará con dos algoritmos diferentes: *SVM* y *Random Forest*, a los cuales iremos cambiando los valores de sus hiperparámetros.

Normalmente, para escoger los valores de los hiperparámetros que mejor funcionan en un problema, se realiza una separación del conjunto de datos en un conjunto de entrenamiento (90% del total) y en un conjunto de test (10% del total). De esta forma, el algoritmo aprenderá con un conjunto grande de datos y podremos comprobar si funciona con un conjunto menor de datos que no ha visto antes.

Por otra parte, también realizaremos "validación cruzada" en cada experimento. La validación cruzada consiste en dividir el conjunto de entrenamiento en *K* subconjuntos (en nuestro caso serán 5 para no aumentar demasiado el tiempo de computación), de tal forma que uno de los subconjuntos sea de validación y el resto de entrenamiento. Este proceso se repite *K* veces, donde en cada una de ellas se cambia el subconjunto de test por uno que anteriormente era de entrenamiento. De esta forma, se emplean diferentes conjuntos para validar y se realiza al final la media de las *K* iteraciones para determinar el rendimiento del modelo, haciendo éste menos sensible a la subpartición del conjunto de datos de entrenamiento [40].

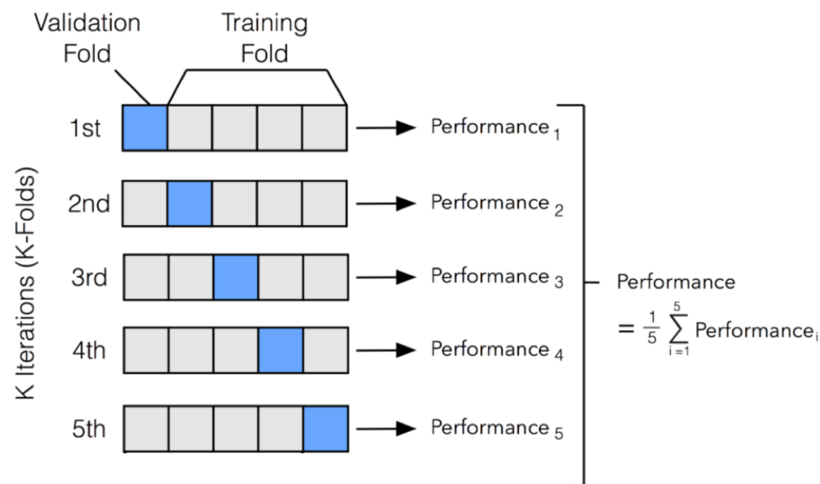

Figura 3-29. Validación cruzada.

3.5.2.1 Support Vector Machines

En esta experimentación se utilizará el módulo *SVM* de la librería *Scikit-learn*, el cual contiene diferentes implementaciones de máquinas de soporte vectorial. En concreto, se utilizará la clase *SVC*, correspondiente a un clasificador de soporte vectorial.

Para realizar el entrenamiento de este algoritmo, probaremos diferentes valores de los siguientes hiperparámetros que posee:

- **C:** este parámetro añade una penalización por cada punto de datos mal clasificado. Si *C* es pequeño, la penalización por puntos mal clasificados es baja, por lo que se elige un límite de decisión con un gran margen a costa de un mayor número de clasificaciones erróneas. Es decir, se trata de una función más simple, pero con menor precisión. Si *C* es grande, la SVM intenta minimizar el número de ejemplos mal clasificados debido a la elevada penalización, lo que da lugar a un límite de decisión con un margen menor. Es decir, una función más compleja que se ajusta mejor a los datos.

  En la Figura 3-30, podemos observar la influencia del parámetro *C* en un ejemplo. Para $C = 1000$, los vectores de soporte que se escogen (indicados con un círculo) dan lugar a un margen prácticamente nulo (coincide con el hiperplano de separación), con lo cual se está realizando un mayor ajuste de la función a los datos, pudiendo llegar a ser muy sensible al ruido en los mismos. Si se disminuye *C*, vemos que se escogen más puntos como vectores de soporte y el margen va creciendo. En este caso, estamos eliminando el "sobreajuste" de la función a los datos, aunque pueden producirse clasificaciones erróneas, al haber considerado un margen mayor.

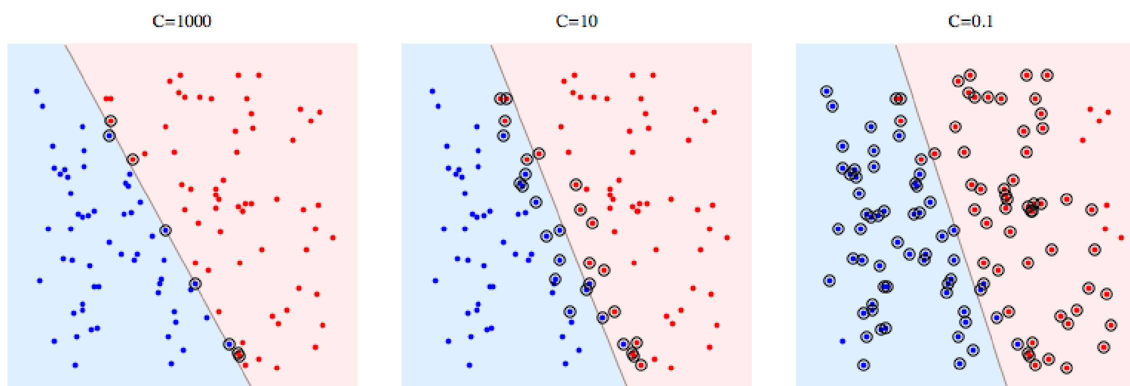

Figura 3-30. Influencia del hiperparámetro *C* en el algoritmo *SVM*.

- **kernel:** mapea las observaciones a otro espacio de características. Lo ideal es que las observaciones sean más fácilmente separables (linealmente) después de esta transformación. Existen múltiples “kernels” o núcleos estándar para estas transformaciones, por ejemplo, el núcleo lineal y el núcleo radial. La elección del núcleo y sus hiperparámetros afectan en gran medida a la separabilidad de las clases (en la clasificación) y al rendimiento del algoritmo.

  En la Figura 3-31, podemos observar la influencia del parámetro *kernel* en un ejemplo. Si utilizamos el *kernel* lineal, observamos que las clases se separan a través de líneas rectas (tal y como su nombre indica). Sin embargo, si utilizamos el *kernel* “RBF”, esta separación entre clases se realiza mediante curvas. Es decir, dependiendo del *kernel* elegido, se optará por separar las clases de una forma u otra.

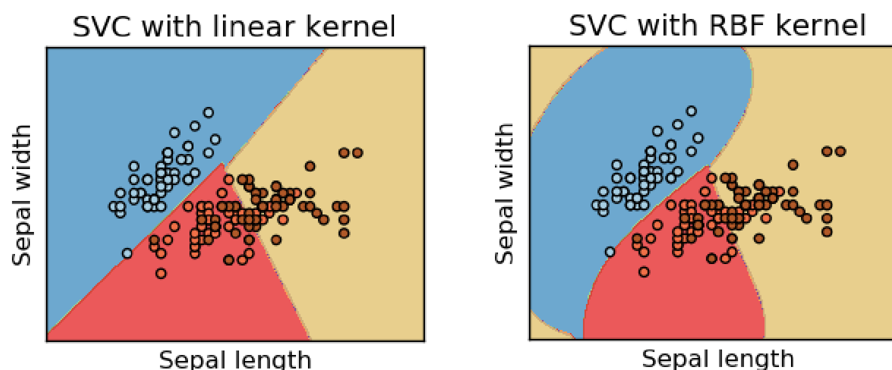

Figura 3-31. Influencia del hiperparámetro *kernel* en el algoritmo *SVM*.

- **gamma:** controla la distancia de influencia de un único punto de entrenamiento. Los valores bajos de *gamma* indican un radio de similitud grande que hace que se agrupen más puntos. Para valores altos de *gamma*, los puntos deben estar muy cerca unos de otros para ser considerados en el mismo grupo (o clase). Este hiperparámetro sólo se configura para los kernel “rbf”, “poly” y “sigmoid”.

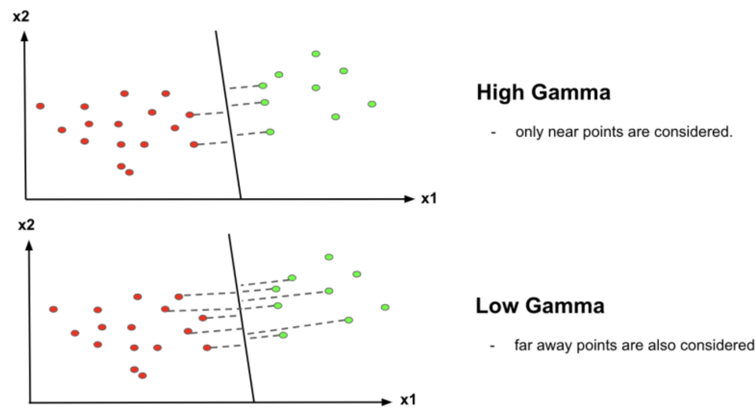

Figura 3-32. Influencia del hiperparámetro *gamma* en el algoritmo *SVM*.

En la siguiente tabla, se muestran los diferentes valores de hiperparámetros con los que se experimentará:

Hiperparámetro	**Valores para experimentar**
C	0.01 | 0.1 | 1 | 10 | 50
kernel	rbf | linear
gamma	0.01 | 1 | 5

Tabla 3-4. Valores de los hiperparámetros con los que se experimentará en *SVM*.

3.5.2.2 Random Forest

La otra experimentación a realizar es con el algoritmo *Random Forest*. Utilizaremos, para ello, la clase *RandomForestClassifier* del módulo *ensemble* de la librería *Scikit-learn*.

Para realizar el entrenamiento de este algoritmo, probaremos diferentes valores de los siguientes hiperparámetros que posee:

- **n_estimators:** el número de árboles en el bosque.
- **criterion:** función para medir la calidad de una división. Los criterios admitidos son "gini" para la impureza de Gini y "entropía" para la ganancia de información.

Impurity	Task	Formula	Description
Gini impurity	Classification	$\sum_{i=1}^{C} f_i(1 - f_i)$	f_i is the frequency of label *i* at a node and *C* is the number of unique labels.
Entropy	Classification	$\sum_{i=1}^{C} -f_i \log(f_i)$	f_i is the frequency of label *i* at a node and *C* is the number of unique labels.

Tabla 3-5. Fórmulas matemáticas del hiperparámetro *criterion* de *Random Forest* para medir la calidad de una división.

- **max_features:** número de características a tener en cuenta a la hora de buscar la mejor partición.

En la siguiente tabla, se muestran los diferentes valores de hiperparámetros con los que se experimentará:

Hiperparámetro	**Valores para experimentar**
n_estimators	50 | 100 | 200
criterion	gini | entropy
max_features	auto | log2

Tabla 3-6. Valores de los hiperparámetros con los que se experimentará en *Random Forest*.

3.5.3 GridSearchCV

Para llevar a cabo todos los experimentos descritos en los dos apartados anteriores, vamos a utilizar la clase *GridSearchCV* de *Scikit-learn* [41]. Esta clase nos ayuda a obtener el rendimiento de cada experimento realizando todos los modelos posibles con los valores de los hiperparámetros que hemos seleccionado [42].

En el código 3-24, se define un grid de parámetros a probar con todos los valores que hemos seleccionado en la tabla 3-4 para experimentar. También se definen las métricas a obtener de cada modelo (las cuales se explicarán posteriormente). Por último, se entrenan todos los posibles modelos, realizando para cada uno de ellos la validación cruzada ($K = cv = 5$).

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score, f1_score, roc_auc_score

# Define el grid de parametros a probar
param_grid = [{'kernel': ['rbf'], 'gamma': [0.01, 1, 5], 'C': [0.01, 0.1, 1, 10,
50]},
              {'kernel': ['linear'], 'C': [0.01, 0.1, 1, 10, 50]}]

# Define las metricas a obtener
scoring = {'accuracy': make_scorer(accuracy_score),
           'f1': make_scorer(f1_score, pos_label=1),
           'roc_auc': make_scorer(roc_auc_score)}

# Entrena el modelo
gs_svc = GridSearchCV(SVC(random_state=0), param_grid, scoring=scoring,
refit='accuracy', cv=5, verbose=10, n_jobs=-1)
gs_svc.fit(X_train, y_train)
```

Código 3-23. *GridSearchCV* para *SVM*.

4 EVALUACIÓN DE RESULTADOS

En este capítulo se muestran los resultados que se han obtenido en los diferentes experimentos descritos en el capítulo anterior y se seleccionarán los mejores modelos en base a esos resultados.

4.1 Métricas de evaluación

Para poder escoger los mejores modelos, necesitamos definir unas métricas para medir el rendimiento de cada uno de ellos y podamos conocer lo bueno o malo que son. Las métricas que se han decidido seleccionar son:

- **Accuracy (exactitud):** es la más sencilla de todas las métricas de evaluación y la más utilizada. Se calcula dividiendo el número de predicciones correctas entre el número total de predicciones. Aunque puede darnos una idea rápida del rendimiento de un clasificador, se utiliza mejor cuando el número de ejemplos en cada clase es aproximadamente equivalente (como en nuestro caso). Cuanto más cerca esté del valor 1, más exactitud tiene el modelo.

$$\boldsymbol{exactitud = \frac{n^{\underline{o}}\ predicciones\ correctas}{n^{\underline{o}}\ total\ predicciones} = \frac{TP + TN}{TP + TN + FP + FN}}$$

- **F1-score:** es una media de otras dos métricas: precisión (precision) y exhaustividad (recall). La precisión es el porcentaje de observaciones etiquetadas por el modelo como clase positiva que realmente pertenecen a la clase positiva (verdaderos positivos frente a falsos positivos). La exhaustividad compara el número de observaciones que el modelo etiquetó como clase positiva con el número total de observaciones de la clase positiva. Alcanza su valor óptimo (1) sólo si la precisión y la exhaustividad son ambas del 100%. Y si una de ellas es igual a 0, entonces también la puntuación F1 tiene su peor valor (0).

$$\boldsymbol{F1 = 2 * \frac{precisi\acute{o}n * exhaustividad}{precisi\acute{o}n + exhaustividad}}$$

$$\boldsymbol{precisi\acute{o}n = \frac{TP}{TP + FP}}$$

$$\boldsymbol{exhaustividad = \frac{TP}{TP + FN}}$$

- **ROC-AUC (área bajo la curva de característica operativa del receptor):** se trata de una métrica que sólo se utiliza para los problemas de clasificación binaria. El área bajo la curva representa la proporción de verdaderos positivos frente a la proporción de falsos positivos. Un valor de 1 en toda el área que cae bajo la curva representa un clasificador perfecto. Esto significa que un AUC de 0.5 es básicamente tan bueno como adivinar al azar.

- **Matriz de confusión:** tabla que representa la precisión de un modelo con respecto a dos o más clases. Las predicciones del modelo estarán en el eje X, mientras que las observaciones se sitúan en el eje Y. Las celdas se rellenan con el número de predicciones que hace el modelo. Las predicciones correctas se encuentran en una línea diagonal que se desplaza de la parte superior izquierda a la inferior derecha.

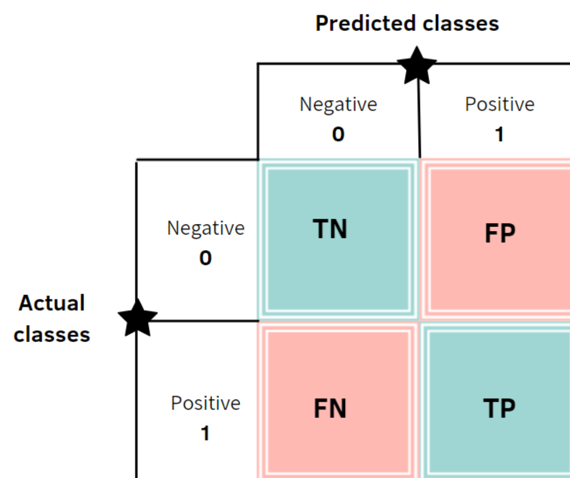

Figura 4-1. Estructura de la matriz de confusión.

En ella se definen las siguientes celdas:

- **True Negative (TN) - Verdaderos Negativos:** cantidad de negativos que fueron clasificados correctamente como negativos en el modelo.
- **False Positive (FP) – Falsos Positivos:** cantidad de negativos que fueron clasificados incorrectamente como positivos en el modelo.
- **False Negative (FN) – Falsos Negativos:** cantidad de positivos que fueron clasificados incorrectamente como negativos en el modelo.
- **True Positive (TP) – Verdaderos Positivos:** cantidad de positivos que fueron clasificados correctamente como positivos en el modelo.

Todas las métricas podemos obtenerlas desde el módulo *metrics* de la librería *Scikit-learn*.

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
```

Código 4-1. Métricas *Accuracy*, *F1-score* y *ROC-AUC*.

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

def ConfusionMatrix(y_pred, y_test):
    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negativo','Positivo']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten() /
np.sum(cf_matrix)]

    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
                xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted label", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("True label"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Matriz de confusion", fontdict = {'size':18}, pad = 20)
```

Código 4-2. Matriz de confusión.

4.2 Evaluación de los modelos SVM

En este apartado procedemos a evaluar los resultados de la experimentación con el modelo *SVM* con los hiperparámetros descritos en la Tabla 3-4.

4.2.1 Idioma inglés

Este experimento se realiza con un conjunto de 50.000 tweets en inglés (25.000 con polaridad negativa y 25.000 con polaridad positiva), a los cuales se les ha realizado *stemming* y se han calculado la media de sus vectores de palabras con *Word2Vec*. Nótese que los valores de las diferentes métricas se han obtenido tras haber realizado validación cruzada.

Hiperparámetros			**Resultados**		
C	**kernel**	**gamma**	**Accuracy**	**F1-score**	**ROC-AUC**
0.01	rbf	0.01	0.50055556	0.66716031	0.5
0.01	rbf	1	0.6864	0.68088906	0.68642001
0.01	rbf	5	0.69553333	0.69557359	0.69553402
0.1	rbf	0.01	0.66788889	0.65902386	0.66791858
0.1	rbf	1	0.70526667	0.70597288	0.7052648

0.1	rbf	5	0.71375556	0.71906353	0.71373542
1	rbf	0.01	0.68753333	0.68291512	0.68755036
1	rbf	1	0.7154	0.71999068	0.71538264
1	rbf	5	0.72204444	0.72731513	0.72202385
10	rbf	0.01	0.69982222	0.69972119	0.69982343
10	rbf	1	0.72255556	0.72766171	0.72253558
10	rbf	5	0.71693333	0.72132817	0.71691669
50	rbf	0.01	0.70593333	0.70767679	0.70592756
50	rbf	1	0.72655556	0.73125933	0.72653692
50	rbf	5	0.69902222	0.70119656	0.69901499
0.01	linear	-	0.68215556	0.67529263	0.68217988
0.1	linear	-	0.69682222	0.69572714	0.69682706
1	linear	-	0.70435556	0.70673478	0.70434739
10	linear	-	0.71235556	0.71611162	0.71234172
50	linear	-	0.71546667	0.71999121	0.71544957

Tabla 4-1. Resultados experimentación *SVM* en el idioma inglés.

Si analizamos lo que hemos obtenido, observamos que se consiguen valores similares de métricas para los diferentes valores de hiperparámetros. Los valores más altos se logran cuando *C* y *gamma* aumentan. Los valores F1 son muy similares a los de la exactitud, lo cual nos indica que la precisión y la exhaustividad de los modelos están equilibradas. Por otro lado, los valores del área bajo la curva rondan el 0.7, lo que nos indica que se discriminan los tweets de una clase y de otra en un 70%.

Podemos apreciar que se consigue la mayor exactitud (0.7266) con los hiperparámetros: *C* = 50, *kernel* = 'rbf' y *gamma* = 1. Este modelo es el que se ha considerado como mejor en este conjunto de experimentos. Si utilizamos este modelo para comprobar su exactitud con nuestros datos de test, los cuales eran un 10% del total, obtenemos lo siguiente:

```
clf = gs_svc.best_estimator_
print('Test Accuracy (SVC): %.3f' % clf.score(X_test, y_test))
```

```
Test Accuracy (SVC): 0.731
```

Figura 4-2. Exactitud del mejor modelo de *SVM* con los datos de test del idioma inglés.

Es decir, hemos conseguido clasificar los tweets con un 73.1% de exactitud.

Si observamos la matriz de confusión del mejor modelo para los datos de test (Figura 4-3), podemos ver que en la mayoría de los casos no se confunden las clases entre sí.

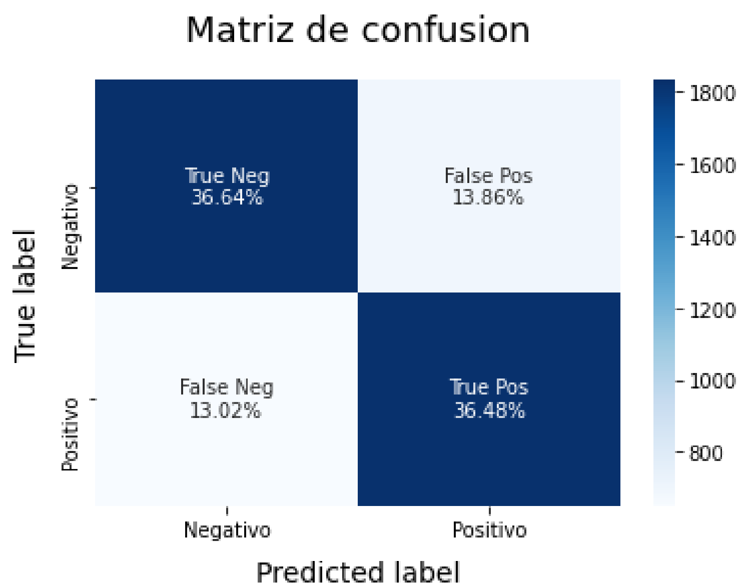

Figura 4-3. Matriz de confusión del mejor modelo de *SVM* con los datos de test del idioma inglés.

4.2.2 Idioma español

Este experimento se realiza con un conjunto de 13.440 tweets en español (aproximadamente la mitad con polaridad positiva y la otra mitad con negativa), a los cuales se les ha realizado *stemming* y se han calculado la media de sus vectores de palabras con *Word2Vec*. Nótese que los valores de las diferentes métricas se han obtenido tras haber realizado validación cruzada.

Hiperparámetros			**Resultados**		
C	**kernel**	**gamma**	**Accuracy**	**F1-score**	**ROC-AUC**
0.01	rbf	0.01	0.51033399	0	0.5
0.01	rbf	1	0.62541321	0.4610222	0.61937582
0.01	rbf	5	0.64045958	0.51409994	0.63536119
0.1	rbf	0.01	0.51033399	0	0.5
0.1	rbf	1	0.64880919	0.52727035	0.64377783
0.1	rbf	5	0.66468246	0.57748268	0.6607039
1	rbf	0.01	0.62632278	0.46711118	0.62041386
1	rbf	1	0.66765831	0.5852486	0.66384213
1	rbf	5	0.67476831	0.60475464	0.67140157
10	rbf	0.01	0.65484426	0.54064626	0.64999151

10	rbf	1	0.67931534	0.61272498	0.67605578
10	rbf	5	0.69006269	0.63596582	0.68729346
50	rbf	0.01	0.66848483	0.58001564	0.66442501
50	rbf	1	0.69072388	0.63420632	0.68783198
50	rbf	5	0.70403425	0.65951308	0.70163777
0.01	linear	-	0.59713921	0.34664003	0.58946898
0.1	linear	-	0.64756931	0.52941057	0.64266098
1	linear	-	0.66865011	0.58021786	0.66459037
10	linear	-	0.67733108	0.60448385	0.67381825
50	linear	-	0.68749948	0.6240974	0.68431051

Tabla 4-2. Resultados experimentación *SVM* en el idioma español.

Si analizamos lo que hemos obtenido, observamos que se consiguen valores similares de métricas que en el caso del idioma inglés, siendo en este caso un poco peores.

Podemos apreciar que se consigue la mayor exactitud (0.7040) con los hiperparámetros: $C = 50$, *kernel* = 'rbf' y *gamma* = 5. Este modelo es el que se ha considerado como mejor en este conjunto de experimentos. Si utilizamos este modelo para comprobar su exactitud con nuestros datos de test, los cuales eran un 10% del total, obtenemos lo siguiente:

```
clf = gs_svc.best_estimator_
print('Test Accuracy (SVC): %.3f' % clf.score(X_test, y_test))

Test Accuracy (SVC): 0.699
```

Figura 4-4. Exactitud del mejor modelo de *SVM* con los datos de test del idioma español.

Es decir, hemos conseguido clasificar los tweets con un 69.9% de exactitud.

Si observamos la matriz de confusión del mejor modelo para los datos de test (Figura 4-5), podemos ver que en este caso se acierta más con la clase negativa (aumentan los Verdaderos Negativos y disminuyen los Falsos Positivos) que en el idioma inglés. Sin embargo, hay más errores con la clase positiva (aumentan los Falsos Negativos y disminuyen los Verdaderos Positivos).

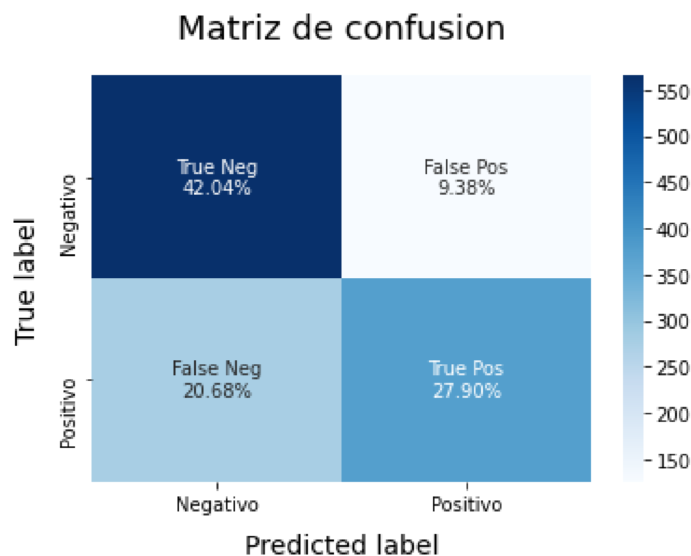

Figura 4-5. Matriz de confusión del mejor modelo de *SVM* con los datos de test del idioma español.

4.3 Evaluación de los modelos Random Forest

En este apartado procedemos a evaluar los resultados de la experimentación con el modelo *Random Forest* con los hiperparámetros descritos en la Tabla 3-6.

4.3.1 Idioma inglés

Este experimento se realiza con un conjunto de 50.000 tweets en inglés (25.000 con polaridad negativa y 25.000 con polaridad positiva), a los cuales se les ha realizado *stemming* y se han calculado la media de sus vectores de palabras con *Word2Vec*. Nótese que los valores de las diferentes métricas se han obtenido tras haber realizado validación cruzada.

Hiperparámetros			**Resultados**		
criterion	**max_features**	**n_estimators**	**Accuracy**	**F1-score**	**ROC-AUC**
gini	auto	50	0.70148889	0.69821965	0.70150178
gini	auto	100	0.70684444	0.70652955	0.7068465
gini	auto	200	0.7084	0.70886541	0.7083991
gini	log2	50	0.69906667	0.69499746	0.69908234
gini	log2	100	0.70526667	0.70395702	0.7052724
gini	log2	200	0.70826667	0.70799013	0.70826858
entropy	auto	50	0.70117778	0.69624591	0.70119664
entropy	auto	100	0.7066	0.70432602	0.70660939

entropy	auto	200	0.70953333	0.70858557	0.70953779
entropy	log2	50	0.70182222	0.69600721	0.70184427
entropy	log2	100	0.70608889	0.70368039	0.7060987
entropy	log2	200	0.70815556	0.70717053	0.70816011

Tabla 4-3. Resultados experimentación *Random Forest* en el idioma inglés.

Si analizamos lo que hemos obtenido, observamos que se consiguen valores similares de métricas que en el caso del algoritmo *SVM*.

Podemos apreciar que se consigue la mayor exactitud (0.7095) con los hiperparámetros: *criterion* = 'entropy', *max_feaures* = 'auto' y *n_estimators* = 200. Este modelo es el que se ha considerado como mejor en este conjunto de experimentos. Si utilizamos este modelo para comprobar su exactitud con nuestros datos de test, los cuales eran un 10% del total, obtenemos lo siguiente:

```
clf = gs_rf.best_estimator_
print('Test Accuracy (Random Forest): %.3f' % clf.score(X_test, y_test))
```

```
Test Accuracy (Random Forest): 0.710
```

Figura 4-6. Exactitud del mejor modelo de *Random Forest* con los datos de test del idioma inglés.

Es decir, hemos conseguido clasificar los tweets con un 71% de exactitud, peor resultado que en el caso del algoritmo *SVM*.

Si observamos la matriz de confusión del mejor modelo para los datos de test (Figura 4-7), podemos ver que en este caso se acierta de forma similar la clase negativa que en el algoritmo *SVM*. Sin embargo, hay más errores con la clase positiva (aumentan los Falsos Negativos y disminuyen los Verdaderos Positivos).

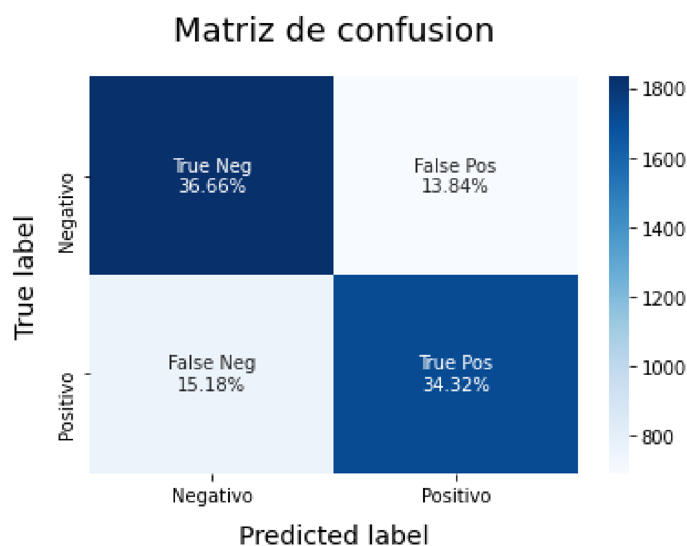

Figura 4-7. Matriz de confusión del mejor modelo de *Random Forest* con los datos de test del idioma inglés.

4.3.2 Idioma español

Este experimento se realiza con un conjunto de 13.440 tweets en español (aproximadamente la mitad con polaridad positiva y la otra mitad con negativa), a los cuales se les ha realizado *stemming* y se han calculado la media de sus vectores de palabras con *Word2Vec*. Nótese que los valores de las diferentes métricas se han obtenido tras haber realizado validación cruzada.

Hiperparámetros			**Resultados**		
criterion	**max_features**	**n_estimators**	**Accuracy**	**F1-score**	**ROC-AUC**
gini	auto	50	0.74768458	0.72580002	0.74634775
gini	auto	100	0.75190059	0.73050696	0.75057801
gini	auto	200	0.75479448	0.73476403	0.75355369
gini	log2	50	0.74975231	0.72668279	0.74832614
gini	log2	100	0.75264572	0.72986553	0.75121923
gini	log2	200	0.75512574	0.73362236	0.75377538
entropy	auto	50	0.75214859	0.73024527	0.75078983
entropy	auto	100	0.75669579	0.73515625	0.75533869
entropy	auto	200	0.755621	0.73435645	0.75429112
entropy	log2	50	0.75132218	0.72834177	0.74989147
entropy	log2	100	0.75281067	0.72991524	0.75137691
entropy	log2	200	0.75504252	0.7328936	0.75364967

Tabla 4-4. Resultados experimentación *Random Forest* en el idioma español.

Si analizamos lo que hemos obtenido, observamos que se consiguen valores mucho mejores que en el caso del algoritmo *SVM*. Incluso son mejores que en el caso del idioma inglés.

Podemos apreciar que se consigue la mayor exactitud (0.7567) con los hiperparámetros: *criterion* = ‘entropy’, *max_feaures* = ‘auto’ y *n_estimators* = 100. Este modelo es el que se ha considerado como mejor en este conjunto de experimentos. Si utilizamos este modelo para comprobar su exactitud con nuestros datos de test, los cuales eran un 10% del total, obtenemos lo siguiente:

```
clf = gs_rf.best_estimator_
print('Test Accuracy (Random Forest): %.3f' % clf.score(X_test, y_test))

Test Accuracy (Random Forest): 0.765
```

Figura 4-8. Exactitud del mejor modelo de *Random Forest* con los datos de test del idioma español.

Es decir, hemos conseguido clasificar los tweets con un 76.5% de exactitud, mejor resultado que en el caso del

algoritmo *SVM*.

Si observamos la matriz de confusión del mejor modelo para los datos de test (Figura 4-9), podemos ver que en este caso se mejora tanto la predicción de la clase negativa (aumentan los Verdaderos Negativos y disminuyen los Falsos Positivos) como la predicción de la clase positiva (aumentan los Verdaderos Positivos y disminuyen los Falsos Negativos) que en el algoritmo *SVM*.

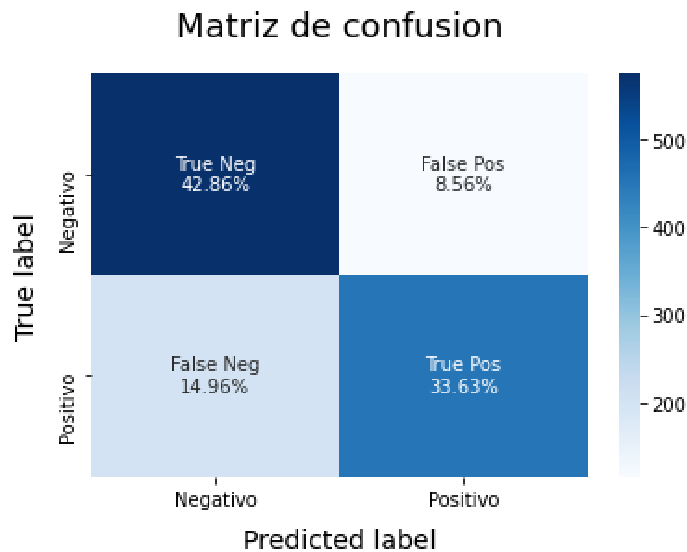

Figura 4-9. Matriz de confusión del mejor modelo de *Random Forest* con los datos de test del idioma español.

4.4 Muestra de tweets clasificados

Una vez analizados los resultados obtenidos para cada uno de los experimentos realizados, vamos a analizar una muestra de tweets que han sido clasificados correctamente y otros que no lo han sido. Para ello, vamos a partir de los tweets utilizados como test y de los mejores modelos obtenidos, tanto para el caso inglés (*SVM* con hiperparámetros *C* = 50, *kernel* = 'rbf' y *gamma* = 1) como para el caso español (*Random Forest* con hiperparámetros *criterion* = 'entropy', *max_feaures* = 'auto' y *n_estimators* = 100).

Una muestra de tweets correctamente clasificados es la siguiente:

@AlexBellamyLeto cuando queráis os llevamos a un gran lugar para comerlos a buen precio (**Positivo**)

🌼 severus snape 🌼\n\n-pobresito \n-nunca le odié me parecía muy estricto e injusto pero sabía que algo pasaba... (**Negativo**)

@marianorajoy saldremos perdiendo no? saltamos de la sarten para caer a las brasas (**Negativo**)

@cuervotinelli que lindo es cuándo me haces reír me haces muy feliz;3 QUE SUERTE QUE TENGO EN TENER UN VIEJO TAN COPADO !!!!!! (**Positivo**)

I burned my hand so haaaard today when i was makin a pizza :/ aui (**Negativo**)

Night time is always fun!!!! have a great sleep people im going to sleep soon i'm tired So goodnight peoplez (**Positivo**)

Figura 4-10. Muestra de tweets correctamente clasificados.

Podemos observar en la Figura 4-10 que en los tweets positivos aparecen palabras como "querer", "gran", "buen", "lindo", "reír", "feliz", "suerte" y "divertido". Estas palabras están asociadas a sentimientos positivos

en la mayoría de los casos y, como podemos ver, el algoritmo es capaz de aprender esto y clasifica esos tweets como positivos.

Por el contrario, en los tweets negativos podemos ver que aparecen palabras como “odiar”, “estricto”, “injusto”, “perder”, “caer” y “quemar”. Estas palabras están asociadas a sentimientos negativos y, en este caso, el algoritmo también lo ha aprendido.

Hay que recordar que hemos utilizado la media de los vectores de palabras obtenidos mediante *Word2Vec* para la creación de los vectores de tweets. Es decir, realmente no estamos teniendo en cuenta estas palabras individualmente, sino el conjunto de todas las existentes en un tweet. Por lo tanto, el algoritmo está aprendiendo que toda esa frase negativa o positiva es la que se considera finalmente para realizar el entrenamiento y, posteriormente, la clasificación de los tweets.

En el caso de la muestra de tweets incorrectamente clasificados tenemos lo siguiente:

Sin vosotros no soy nada. Con la fuerza del sur ganaremos el futuro. Nadie nos va a poner de rodillas. (**Positivo y clasificado como negativo**)

Mi derecho y obligación ejercido!! Eso si por favor optimizen y faciliten el proceso de voto por correo. #elecciones #20N (**Positivo y clasificado como negativo**)

Sevillanas Rihanna....el mitin de @marianorajoy es una fiesta. Montoro entra y saluda al público y a la prensa. #elecciones2011 #20N (**Negativo y clasificado como positivo**)

@periodistas21 Confío en que la lean Dívar y el CGPJ y se lo apliquen cuando se critiquen las sentencias del Supremo y a sus magistrados (**Positivo y clasificado como negativo**)

Basagoiti: no asumen q somos mejores. Pues q les den... #Guiñoles (**Negativo y clasificado como positivo**)

Looking for @AlRocker this morning ... where are ya, doll? I got some cuerstions for you .... (**Positivo y clasificado como negativo**)

Good afternoon everyone, just woke up with no plans to do anything (**Positivo y clasificado como negativo**)

Figura 4-11. Muestra de tweets incorrectamente clasificados.

En la Figura 4-11 podemos destacar que, al igual que antes, hay palabras que están asociadas la mayoría de las veces a tweets positivos (“fiesta”, “saludar”, “mejor”) y otras asociadas a negativos (“criticar”). En estos casos, el algoritmo clasifica los tweets en los que aparecen estas palabras como tales, pero realmente no corresponden a esos sentimientos otorgados originalmente, sino a los contrarios. Esto puede deberse a la ironía de las frases como, por ejemplo, ocurre en el tercer tweet de la Figura 4-11: “el mitin de @marianorajoy es una fiesta. Montoro entra y saluda al público y a la prensa”. Ese mitin debería tratarse como algo serio y el autor del tweet comenta que ahora mismo es una “fiesta”, como algo desorganizado. Es por ello por lo que el tweet originalmente tiene una polaridad negativa, se trata de una ironía. Esto es lo que se comentó anteriormente en la sección 3.3.1.1: lo difícil de que una máquina pueda distinguir si algo causa sentimiento negativo o positivo reside en la capacidad de entender la ironía o el sarcasmo en las oraciones.

Por otra parte, también hay tweets en la base de datos etiquetados manualmente que no tienen realmente una polaridad negativa o positiva, sino que son neutrales. Y quizás a las máquinas, al igual que a los humanos, les cueste “decidir” entre dos polaridades.

5 CONCLUSIONES Y LÍNEAS DE CONTINUACIÓN

En este trabajo hemos podido estudiar e implementar dos técnicas de Machine Learning para llevar a cabo el análisis de sentimientos en la red social de Twitter. Se ha visto la teoría que hay detrás de estos dos algoritmos, los cuales trabajan y aprenden de forma diferente.

En líneas generales del trabajo, hemos profundizado sobre cómo funcionan los métodos de clasificación automática, en concreto nos hemos centrado en dos de los más utilizados (*Support Vector Machines* y *Random Forest*). Tras ello, hemos llevado a cabo una limpieza profunda de los datos de los que partíamos, eliminando mayúsculas, codificaciones, usuarios, hashtags, enlaces, expansiones, tildes, *stopwords* y realizando corrección ortográfica y *stemming*. Después de eso, hemos pasado los tweets a vectores, utilizando dos métodos diferentes: *Word2Vec* y *Doc2Vec*. De esta forma hemos conseguido realizar una extracción de características de las palabras y hemos comprobado que, aunque a priori *Doc2Vec* debería ser mejor al tratarse de una técnica más compleja que *Word2Vec,* hemos visto que no es así. Finalmente, hemos pasado estos vectores a los clasificadores. Estos clasificadores los hemos implementado gracias a los métodos existentes en la librería *Sciki-learn*, la cual nos ha sido muy útil a lo largo del trabajo.

A través de estos clasificadores, hemos logrado obtener diferentes modelos capaces de clasificar los tweets con polaridad positiva y negativa en un 73.1% de exactitud para el caso del idioma inglés, y en un 76.5% para el caso español. De esta forma, hemos podido observar y comparar los distintos resultados que se alcanzan con cada uno de los modelos.

Por otro lado, hay que destacar también que toda la experimentación se ha llevado a cabo realizando validación cruzada. Es decir, todos los modelos obtenidos son generalistas, no dependiendo los mismos de las separaciones entre los datos de entrenamiento y test.

Una de las cosas más importantes cuando nos enfrentamos a un problema de clasificación de texto es el tipo de texto y las palabras que hay dentro de ellos. Como ya hemos comentado anteriormente, no es lo mismo la palabra “Hello” que la palabra “hello”. Esto puede suponer un impacto importante a la hora de extraer las características de las palabras para que los algoritmos de Machine Learning aprendan. Es por ello por lo que el efecto de aplicar transformaciones previas a los datos puede mejorar el rendimiento en estos métodos de clasificación y que todo depende del conjunto de datos utilizado y del tipo de lenguaje que posea. En consecuencia, hay que realizar un análisis de datos, aplicar transformaciones a los mismos y realizar un filtrado sobre aquellos que tienen menos importancia. De esta forma conseguiremos que el algoritmo de aprendizaje automático aprenda más eficientemente y generalice mejor.

Cabe destacar que este preprocesamiento del texto ayuda también a disminuir el tiempo y el trabajo de computación de las máquinas, puesto que hoy en día éstas tienen limitaciones y no pueden manejar todos los datos sin ningún tipo de proceso previo. Aún así, en nuestro caso, hemos tenido que disminuir el número de datos para llevar a cabo los diferentes experimentos, ya que es una tarea que conlleva mucho tiempo y disponibilidad de recursos.

En general, los métodos de aprendizaje automático tienden a dar resultados similares y, de nuevo, los resultados dependen del tipo de datos utilizado en el entrenamiento.

Respecto a las líneas de continuación futuras, se podrían aplicar los modelos obtenidos a otros conjuntos de

datos de otros sitios o redes sociales de reseñas, como por ejemplo Amazon [43] o IMDB [44], pasando a ser reseñas de productos y películas, para así ver cómo se comportan los modelos en otro contexto diferente al entrenado como para su aplicación en otros sitios.

Si se poseen máquinas más potentes para llevar a cabo los experimentos, se puede aumentar la cantidad de datos al realizar el entrenamiento y así, posiblemente, se consigan mejores resultados.

Por otra parte, se podrían probar otros algoritmos de clasificación automática existentes o, incluso, otros métodos de extracción de características de palabras, como GloVe [45] o InferSent [46].

Por último, también se podría llevar a cabo la implementación de una red neuronal para realizar el análisis de sentimientos. Los modelos de aprendizaje profundo [47] cuentan con una mayor cantidad de capas y neuronas, las cuales trabajan entre sí para conseguir combinar diferentes parámetros y encontrar los que mejores resultados proporcionen. De esta forma, podríamos tener una mejor perspectiva sobre el funcionamiento que ofrecen las distintas arquitecturas y, posiblemente, encontrar un modelo que funcione aún mejor.

REFERENCIAS

[1] «Twitter,» [En línea]. Disponible en: https://twitter.com/.

[2] «Facebook,» [En línea]. Disponible en: https://www.facebook.com/.

[3] B. Pang, L. Lee y S. Vaithyanathan, «Thumbs up?: Sentiment classification using machine learning techniques,» *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language,* vol. 10, pp. 79-86, 2002.

[4] S. Russell y P. Norvig, Artificial Intelligence: A Modern Approach, Prentice hall, 2009.

[5] IBM Cloud Education , «Natural Language Processing (NLP),» [En línea]. Disponible en: https://www.ibm.com/cloud/learn/natural-language-processing.

[6] Dave, K, Lawrence, S y Pennock, D. M, «Mining the peanut gallery: Opinion extraction and semantic classification of product reviews,» *Proceedings of the 12th international conference on World Wide Web,* pp. 519-528, 2003.

[7] MonkeyLearn, «Sentiment Analysis: A Definitive Guide,» [En línea]. Disponible en: https://monkeylearn.com/sentiment-analysis/.

[8] S. Raschka y V. Mirjalili, «Applying Machine Learning to Sentiment Analysis - Transforming words into feature vectors,» de *Python Machine Learning (Third Edition)*, Birmingham, Packt, 2019.

[9] C. Rodríguez Abellán, «Word Embeddings: cómo la IA nos muestra la evolución de las palabras,» [En línea]. Disponible en: https://empresas.blogthinkbig.com/word-embedings-como-la-ia-nos-muestra-la-evolucion-de-las-palabras/.

[10] S. Fan, «Understanding Word2Vec and Doc2Vec,» [En línea]. Disponible en: https://shuzhanfan.github.io/2018/08/understanding-word2vec-and-doc2vec/.

[11] N. Bambrick, «Support Vector Machines: A Simple Explanation,» [En línea]. Disponible en: https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html.

[12] z_ai, «Random Forest Explained,» [En línea]. Disponible en: https://towardsdatascience.com/random-forest-explained-7eae084f3ebe.

[13] «Python,» [En línea]. Disponible en: https://www.python.org/.

[14] M. Zaforas, «¿Es Python el lenguaje del futuro?,» [En línea]. Disponible en: https://www.paradigmadigital.com/dev/es-python-el-lenguaje-del-futuro/.

[15] «NLTK,» [En línea]. Disponible en: https://www.nltk.org/.

[16] S. Bird, E. Klein y E. Loper, «Processing Raw Text,» de *Natural Language Processing with Python -*

Analyzing Text with the Natural Language Toolkit.

[17] «Beautiful Soup,» [En línea]. Disponible en: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[18] «Symspellpy,» [En línea]. Disponible en: https://github.com/mammothb/symspellpy.

[19] «Unidecode,» [En línea]. Disponible en: https://pypi.org/project/Unidecode/.

[20] «Wordcloud,» [En línea]. Disponible en: https://pypi.org/project/wordcloud/.

[21] «Gensim,» [En línea]. Disponible en: https://radimrehurek.com/gensim/.

[22] «Scikit-learn,» [En línea]. Disponible en: https://scikit-learn.org/stable/.

[23] «Jupyter,» [En línea]. Disponible en: https://jupyter.org/.

[24] «Google Colab,» [En línea]. Disponible en: https://colab.research.google.com/notebooks/intro.ipynb.

[25] Kazanova, «Sentiment140 dataset with 1.6 million tweets,» [En línea]. Disponible en: https://www.kaggle.com/kazanova/sentiment140.

[26] «TASS: Workshop on Semantic Analysis at SEPLN - Dataset Download,» [En línea]. Disponible en: http://tass.sepln.org/tass_data/download.php.

[27] A. Pak y P. Paroubek, «Twitter as a corpus for sentiment analysis and opinion mining,» *LREc,* vol. 10, 2010.

[28] «Convert XML to CSV in Python,» [En línea]. Disponible en: https://www.geeksforgeeks.org/convert-xml-to-csv-in-python/.

[29] D. Sarkar, «Processing and Understanding Text,» de *Text Analytics with Python (Second Edition)*, Bangalore, Apress, 2019.

[30] «El lenguaje de la ironía es el elemento clave de la reputación online,» [En línea]. Disponible en: http://www.brandchats.com/es/2014/02/26/el-lenguaje-de-la-ironia-es-el-elemento-clave-de-la-reputacion-online/.

[31] «Word2Vec,» [En línea]. Disponible en: https://radimrehurek.com/gensim/models/word2vec.html.

[32] «How word2vec works,» [En línea]. Disponible en: https://code.google.com/archive/p/word2vec/.

[33] «TSNE,» [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html#sklearn.manifold.TSNE.

[34] D. Sarkar, «Feature Engineering for Text Representation - Robust Word2Vec Models with Gensim,» de *Text Analytics with Python (Second Edition)*, Bangalore, Apress, 2019.

[35] N. G, «Twitter Sentiment Analysis - word2vec, doc2vec,» [En línea]. Disponible en: https://www.kaggle.com/nitin194/twitter-sentiment-analysis-word2vec-doc2vec.

[36] «Doc2Vec,» [En línea]. Disponible en: https://radimrehurek.com/gensim/models/doc2vec.html.

[37] D. Nelson, «Overview of Classification Methods in Python with Scikit-Learn,» [En línea]. Disponible en: https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/.

[38] «C-Support Vector Classification,» [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[39] «Random Forest classifier,» [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[40] S. Raschka y V. Mirjalili, «Learning Best Practices for Model Evaluation and Hyperparameter Tuning - K-fold cross-validation,» de *Python Machine Learning*, Birmingham, Packt, 2019.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, «Scikit-learn: Machine learning in Python,» *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[42] M. Konkiewicz, «Grid search for parameter tuning,» [En línea]. Disponible en: https://towardsdatascience.com/grid-search-for-parameter-tunning-3c6ff94e7a25.

[43] «Amazon,» [En línea]. Disponible en: https://www.amazon.es/.

[44] «IMDB,» [En línea]. Disponible en: https://www.imdb.com/.

[45] «GloVe,» [En línea]. Disponible en: https://nlp.stanford.edu/projects/glove/.

[46] «InferSent,» [En línea]. Disponible en: https://github.com/facebookresearch/InferSent.

[47] I. Goodfellow, Y. Bengio y A. Corville, Deep Learning, MIT Press, 2016.