UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Detección de titulares satíricos y humorísticos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Hilario López Sanz

Tutor: Carlos David Martínez Hinarejos

Curso 2019-2020

Resumen

Este proyecto pretende abordar la clasificación de titulares de noticias en función de si estas son satíricas o no. Para ello, se comenzará por la extracción de las muestras mediante *web scraping* en páginas de periódicos. A continuación, se aplicarán técnicas de procesamiento de lenguaje con el objetivo de seleccionar la información más relevante de dichos titulares. Por último, se construirá un clasificador utilizando algunos de los algoritmos de aprendizaje automático existentes.

Palabras clave: Clasificación de texto, aprendizaje automático, procesamiento de lenguaje natural.

Abstract

This project approaches the classification of news headlines based on whether they are satirical or not. To do this, we will begin by extracting the samples by using web scraping in different web pages. After that, language processing techniques will be applied to extract the most relevant information. Lastly, a classifier will be built applying some of the existing machine learning algorithms.

Keywords: Text classification, machine learning, natural language processing.

Tabla de contenidos

Tabla de figuras, ecuaciones y tablas

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Motivación

El aprendizaje automático es un área de la informática que recientemente ha ganado popularidad debido al gran volumen de información disponible. En consecuencia, las empresas realizan una mayor inversión en el tratamiento de los datos con el fin de obtener información relevante y poder ofrecer un mejor producto ajustándolo a las preferencias de su cliente.

Personalmente, este campo me resulta especialmente atractivo dado que prueba cómo tareas aparentemente complejas e imposibles de automatizar se pueden realizar con resultados asombrosos. Es el caso de muchas tareas, desde la clasificación de correos electrónicos [1] para evitar aquellos no deseados hasta realizar diagnósticos médicos [2], reconocer la voz y responder adecuadamente [3] o crear máquinas capaces de vencer al mejor jugador de ajedrez [4].

Sin embargo, este es solo el principio y su aplicación se extiende por todas las áreas mostrando su gran potencial y permitiendo ser una herramienta de apoyo para la toma de decisiones o incluso sustituir al humano por una máquina que realice la misma función de manera mucho más eficiente.

Un posible uso de estas técnicas en la sociedad actual es la discriminación entre noticias auténticas y parodias con finalidad satírica y humorística. Dado el perfil actual de obtención de información, donde la mayor parte de las personas apenas disponen del tiempo de corroborar la fiabilidad de la fuente de un titular, resultaría de gran utilidad disponer de una herramienta que permita realizar dicha discriminación automáticamente.

Además, la realización de este trabajo me resultará de ayuda en la comprensión de los conceptos y cómo estos se implementan en un caso real partiendo completamente de cero. De este modo adquiero un mayor conocimiento de las herramientas utilizadas actualmente y la metodología para resolver problemas similares.

1.2. Objetivos

El objetivo del proyecto consiste en construir un clasificador capaz de diferenciar si un título de una noticia es satírico o no. Sin embargo, se dividirá el objetivo principal en los siguientes objetivos más específicos:

- Extraer automáticamente los titulares de noticias de una o varias páginas web.
- Comparar cómo afectan algunas de las técnicas de procesado de texto previamente a la construcción del clasificador. Se estudiarán las técnicas de eliminación de *stopwords* y *stemming.*
- Comparar distintos métodos para la construcción del clasificador. En particular, el clasificador bayesiano multinomial, las máquinas de vector soporte, el perceptrón multicapa y el modelo Doc2Vec.

1.3. Estructura

A continuación, se documenta la estructura seguida exponiendo los puntos principales, así como una breve descripción de estos.

- Introducción: formada por la motivación que ha llevado a la realización del proyecto y los objetivos a alcanzar.
- Estado del arte: introducción a la situación actual respecto a la tecnología y metodología utilizada para la resolución de problemas similares.
- Desarrollo de la solución: proceso llevado a cabo para la resolución del problema, junto a la comparativa de los diferentes métodos empleados.
- Conclusiones: resumen de la experimentación y los resultados obtenidos, además de la solución propuesta.
- Trabajo futuro: posibles mejoras a la hora de abordar este tipo de problemas, así como nuevas áreas que cubrir.

2. Estado del arte

En este apartado se explicarán los conceptos básicos necesarios para comprender cómo se resuelven este tipo de problemas en la actualidad. Se comenzará por mencionar los pasos a seguir para la resolución de problemas similares, explicando de manera general cada etapa.

En la figura 1 se describen las etapas a seguir, desde la extracción de datos mediante sensores, seguido de la representación de dichos datos en el formato correspondiente, hasta el reconocimiento de una muestra gracias al entrenamiento del clasificador.

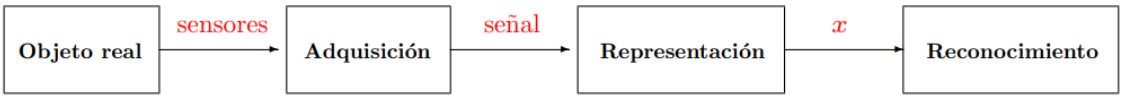

Figura 1: *Etapas de la percepción*

Se parte de un objeto real como puede ser la cara de una persona, el sonido de cierto tipo de ave, o la noticia de un periódico digital. Posteriormente, se deberá adquirir dicho objeto utilizando algún mecanismo, como puede ser una cámara, un micrófono o una herramienta que obtenga información de una página web. Tras este proceso se dispone de lo que se conoce como señal; esto puede ser una imagen de píxeles, un audio o un documento de texto. Finalmente, se hará uso de dicha señal para representarla como un vector. Esta muestra (x) servirá para entrenar el clasificador con el objetivo de reconocer futuras muestras.

Resulta de especial interés centrarse en la etapa de representación de la señal cuando esta se trata de un documento de texto, donde la transformación de texto en un vector no resulta ser una tarea trivial. En la figura 2 se divide dicha tarea de representación en dos partes: preproceso y vectorización.

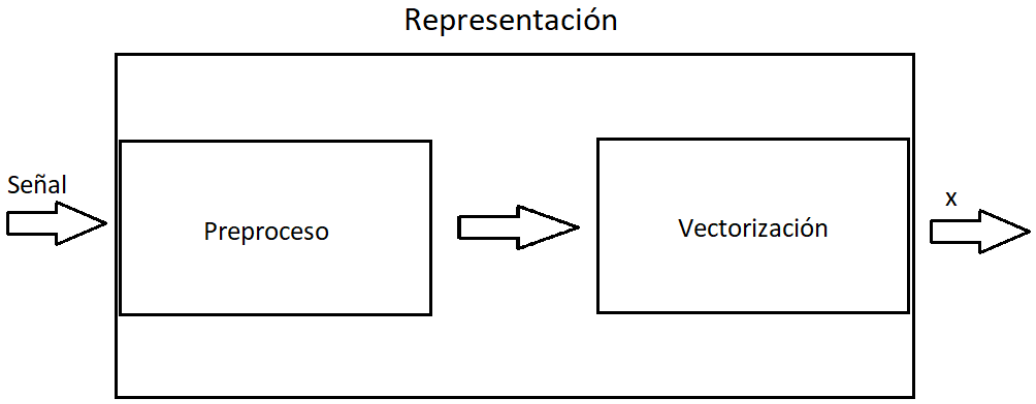

Figura 2: *Representación de la muestra*

En la primera parte donde se realiza el preproceso cabe destacar técnicas como la eliminación de *stopwords*, *stemming* y lematización, de las cuales se explicarán con más detalle las dos primeras. Respecto a la vectorización, el método más común es la representación *bag of words* utilizando como token un unigrama. Esta representación se detallará posteriormente para el caso de unigramas, pero resulta de interés conocer que un token puede también ser una letra o un conjunto de n palabras (n-grama).

Respecto al apartado de reconocimiento, existen múltiples modelos y algoritmos que utilizar a la hora de construir el clasificador, los cuales podemos clasificar en dos grandes grupos: aprendizaje supervisado y aprendizaje no supervisado.

El aprendizaje supervisado se basa en aprender a clasificar una muestra correctamente habiendo visto anteriormente una gran cantidad de muestras que se sabía clasificar. En la figura 3 se nombran algunos de los modelos que existen. Sin embargo, solo se utilizarán tres de ellos, explicados con mayor énfasis posteriormente.

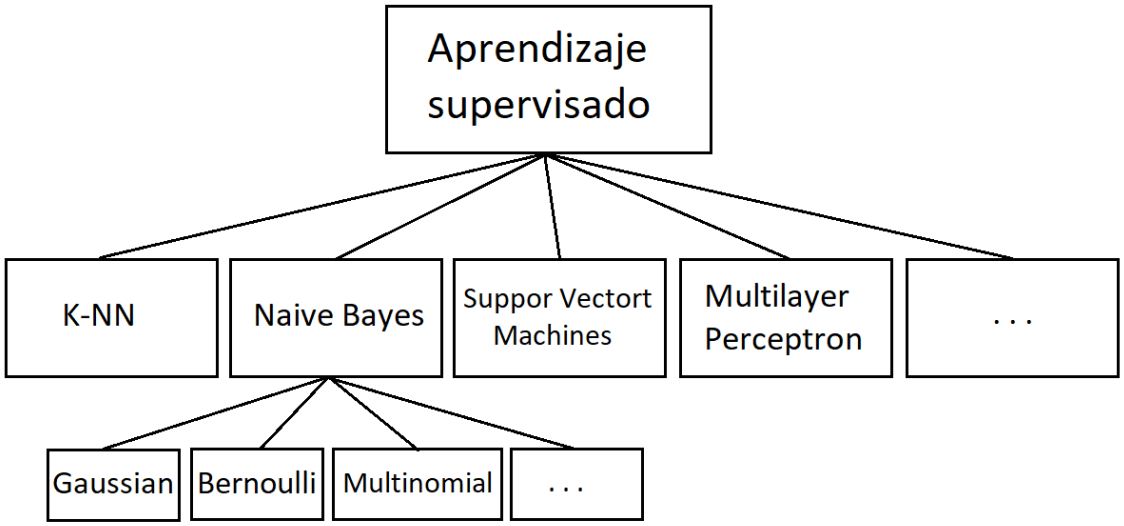

***Figura 3**: Aprendizaje supervisado*

A su vez, existen modelos de aprendizaje no supervisado, es decir, se parte de un conjunto de muestras de las que se desconoce el número de clases diferentes a las que pertenecen y se intenta predecir cuántas clases existen y a qué clase pertenece cada una de las muestras. En la figura 4 se muestran algunos de los modelos más importantes.

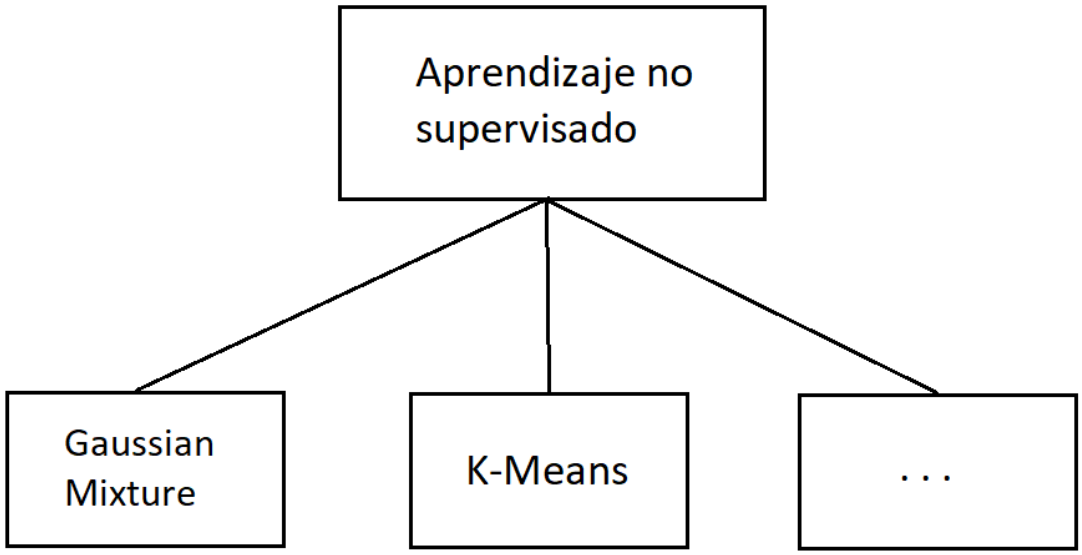

***Figura 4:** Aprendizaje no supervisado*

Partiendo de esta base, se describirán algunas técnicas de procesado de texto, la representación *bag of words*, necesaria para construir los diferentes clasificadores mencionados. Esta representación *bag of words* puede provocar que los vectores resultantes tengan una dimensión muy alta, lo cual tiene ciertas consecuencias negativas en la tarea de clasificación. Es por ello por lo que también se comentará una de las técnicas más utilizadas para la reducción de dimensionalidad de los datos (PCA).

A su vez, se dará una breve descripción del modelo Doc2Vec, modelo que permite realizar dicha clasificación haciendo uso de una representación completamente diferente de las muestras.

2.1. Preprocesado

El preprocesado consiste en la transformación de un texto de modo que se extraiga la información con mayor importancia, eliminando así todo aquello que sea irrelevante y que pueda considerarse ruido. Entre estas técnicas se encuentran la eliminación de *stopwords*, y el *stemming*.

2.1.1. Stopwords

La eliminación de *stopwords* consiste en eliminar aquellas palabras más frecuentes de cierto idioma, dado que no aportan información para la distinción de textos.

Entre estas palabras se encuentran los artículos, pronombres, preposiciones, conjunciones, ciertas formas verbales y verbos auxiliares, entre otros.

2.1.2. Stemming

El *stemming* consiste en la reducción de una palabra a su raíz, de modo que palabras con un mismo lexema y donde solo varía el sufijo serán reducidas a su raíz con el objetivo de compactar la información.

2.2. Bag of words

La representación *bag of words* es uno de los métodos más comunes de representación de documentos dada su fácil implementación y sus buenos resultados a la hora de utilizarse para la construcción de clasificadores.

Dicha representación consta de una matriz NxD, donde el número de filas N corresponde al número de documentos[1] y el número de columnas D a la talla del diccionario, entendiendo como diccionario el conjunto de todos los tokens diferentes que aparecen en todos los documentos (para este caso, se asumirá que un token es una palabra). Dado cierto documento, el valor de la columna i-ésima corresponderá al número de veces que ocurre la palabra i-ésima del diccionario en dicho documento. En la figura 5 se propone un ejemplo con dos documentos junto a la matriz correspondiente.

```
0: La casa es roja
1: La cama es verde
```

	La	casa	es	roja	cama	verde
0	1	1	1	1	0	0
1	1	0	1	0	1	1

Figura 5*:* Ejemplo representación bag of words

2.3. PCA

PCA (*Principal Component Analysis*) [5] es uno de los métodos más utilizados cuando se trata de reducir la dimensionalidad de un conjunto de datos. PCA se utiliza como solución al conocido problema de la maldición de dimensionalidad [6]. Un alto número de dimensiones conlleva que surjan con más frecuencia diversos problemas como la correlación, la existencia de ruido y la ineficiencia del clasificador.

Además, algunos algoritmos como K vecinos más cercanos (K-NN) o K-Means se basan en el cálculo de distancias entre muestras para la clasificación de las mismas, por lo que aumentar las dimensiones provoca que las muestras estén mas separadas entre sí, haciendo que todas se encuentren prácticamente a la misma distancia. Este espacio de grandes

[1] Un documento consiste en un fragmento de texto sin importar la extensión de este; en este caso particular un documento es un titular de noticia.

dimensiones y con muestras separadas se puede solucionar aumentando la cantidad de muestras "rellenando" dicho espacio. Sin embargo, el aumento de muestras debe ser exponencial junto al número de dimensiones, tarea que suele ser imposible en la práctica.

PCA permite solucionar el problema encontrando los *k ejes principales*, es decir, aquellos en los que la varianza que se mantiene tras proyectar los datos sobre dichos ejes es máxima, dicho en otras palabras, nos permite "agrupar" la información que proporcionan las distintas variables de una muestra en un número más reducido de estas manteniendo la máxima información.

2.4. Multinomial Naive Bayes

Este algoritmo de aprendizaje supervisado [7] es usualmente utilizado para la clasificación de muestras linealmente separables. Un claro uso es el de clasificación de texto partiendo de una representación *bag of words.* Este algoritmo asume la independencia de las variables; sin embargo, pese a que esto pueda no ser del todo cierto, en la práctica se obtienen resultados muy satisfactorios [8].

Dada una muestra, será representada mediante un vector de características desde x_1 hasta x_n, donde n representa la talla del diccionario. A partir del teorema de Bayes se puede deducir la siguiente fórmula:

$$\hat{y} = arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Donde $\hat{y}$ representa la clase en la que se clasificará la muestra, $P(y)$ es la probabilidad a priori, es decir, la cantidad de muestras de la clase y dividido por la cantidad de muestras totales y $P(x_i \mid y)$ viene dado por θ_{yi}:

$$\widehat{\theta_{yi}} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Donde N_{yi} define el número de veces que la característica *i* aparece para todas las muestras de la clase *y* y N_y es la suma de todas las características para todas las muestras de la clase *y*. El símbolo α se conoce como suavizado de Laplace [9] y previene que θ_{yi} sea cero, evitando que el productorio de la primera fórmula sea cero.

2.5. Support Vector Machines

Las máquinas de vector soporte (SVM) [10] es uno los métodos de aprendizaje supervisado que permite la clasificación de muestras linealmente separables. Sin embargo, se hace uso del llamado *truco del kernel* [11], lo que permite separar muestras llevándolas a un espacio de mayor dimensionalidad. No obstante, la representación no se llega a producir nunca ya que solo es necesario el valor del producto escalar, obtenido al calcular la función kernel.

Para comprender este concepto se propone un ejemplo con muestras pertenecientes a dos clases diferentes. El objetivo será dibujar una recta que separe las dos clases. En la figura 6 se aprecia como existen varias soluciones, pero se deberá elegir la mejor de ellas.

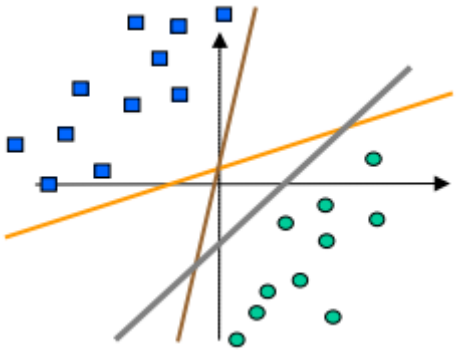

Figura 6: *Posibles fronteras lineales*

Se define como frontera óptima aquella cuyo margen es mayor. En la figura 7 la recta roja representa dicha frontera.

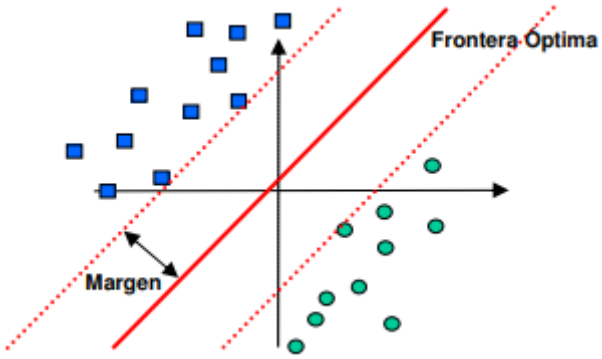

Figura 7: *Frontera lineal óptima*

El problema estaría resuelto si las clases son linealmente separables, esto es, podemos trazar dicha frontera de modo que se consiguen clasificar todas las muestras correctamente. Si bien esto puede suceder, en la práctica no es el caso. En consecuencia, se deberá permitir la clasificación errónea de ciertas muestras, intentando minimizar este error, pero, a su vez, intentando maximizar el margen de la frontera.

Aun así, existen casos en los que no es posible obtener una buena frontera dada la naturaleza de los datos, este es el caso de la figura 8.

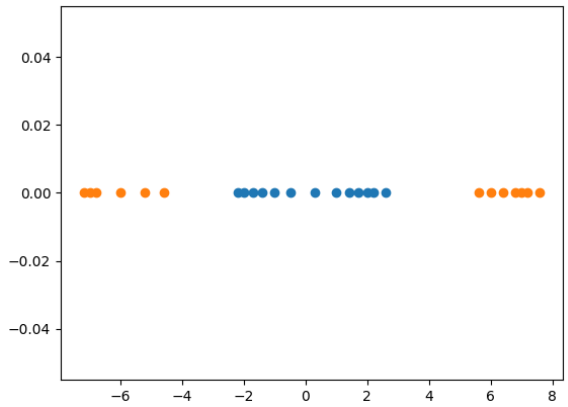

Figura 8: *Frontera lineal imposible*

En este tipo de casos, el uso de una frontera lineal no permite la clasificación de las muestras. Este problema no se daría si se decidiese aumentar la dimensionalidad de los datos [12] a dos dimensiones, donde la segunda dimensión es igual al cuadrado de la primera ($y = x^2$). Para una mejor visualización de lo que está sucediendo se representan las muestras en dos dimensiones en la figura 9.

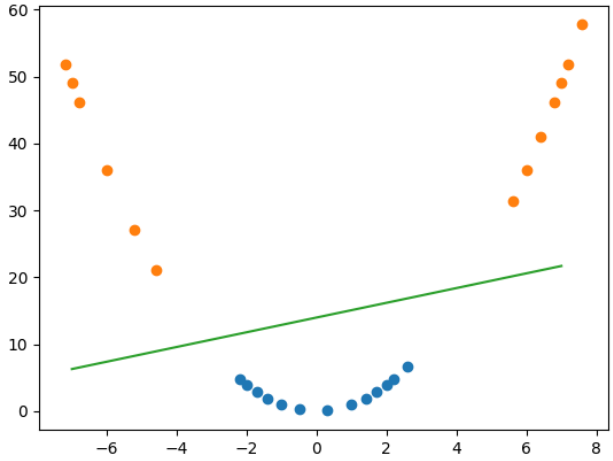

Figura 9: *Aumento de dimensionalidad de los datos*

En esta nueva representación los datos sí son linealmente separables, como muestra la recta verde. Aplicando este mecanismo se consigue hacer posible la clasificación de las muestras partiendo de un caso donde era imposible dicha clasificación.

Esta es la idea de los llamados kernel, funciones que nos permiten llevar los datos a un espacio de mayor dimensionalidad. Cabe destacar que el cálculo del producto escalar en un espacio de dimensión mayor resulta más fácil de calcular que la posición de un punto en dicho espacio y es por esto por lo que no se requiere calcular los puntos explícitamente.

Seguidamente se nombrarán dos tipos de kernel que pueden ser utilizados y resultan de gran interés.

2.5.1. Kernel polinómico

El kernel polinómico corresponde a la fórmula:

$$K(x, y) = (x^T y + c)^d$$

En esta fórmula, tanto *x* como *y* son vectores de características y *d* nos permite generar límites de decisión no lineales cuando su valor es mayor que uno. Para el caso en que *c* es cero y *d* es uno se obtiene el mismo resultado que un kernel lineal. Por el contrario, si se aumenta drásticamente el valor de *d* se produce el problema de *overfitting* [13], donde el clasificador actuará especialmente bien con las muestras utilizadas para la estimación de la frontera y con resultados peores cuando se enfrenta a nuevas muestras que no han sido vistas anteriormente.

Un ejemplo del uso de este kernel se muestra en la figura 10, donde la frontera de decisión deja de ser una recta [14].

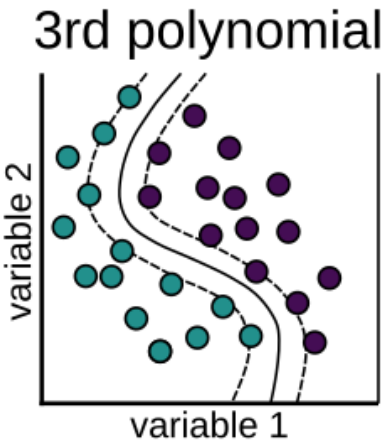

Figura 10: *Ejemplo de kernel polinómico*

2.5.2. Kernel RBF (Radial Basis Function)

El kernel se corresponde a la fórmula:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

En este caso, *x* y *x'* son los vectores implicados, y γ es el parámetro que permite adaptar el clasificador a la "forma" de los datos. Si su valor es muy bajo, no se adaptará correctamente, mientras que si es muy alto se producirán problemas de *overfitting*.

La figura 11 muestra un ejemplo de cómo define la frontera de decisión este tipo de kernel [14].

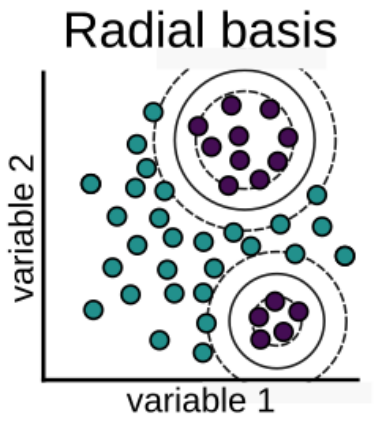

Figura 11: *Ejemplo de kernel RBF*

2.6. Perceptrón multicapa (MLP)

El perceptrón multicapa, también conocido como red neuronal, es una de las técnicas de aprendizaje supervisado capaz de resolver problemas no linealmente separables. En la figura 12 [15] se tiene una representación gráfica del mismo.

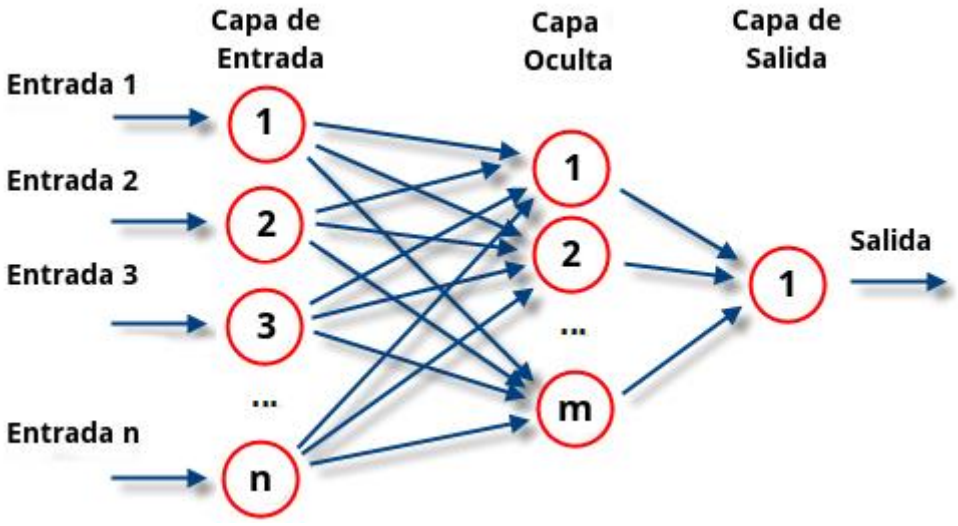

Figura 12: *Perceptrón multicapa*

Dicha red neuronal dispone de una capa de entrada con *n* neuronas, donde *n* es el número de características de las muestras, una o más capas ocultas y una capa de salida. A menudo se hace uso del *one hot encoding* [16], de modo que se tienen tantas neuronas de salida como clases diferentes en los datos a clasificar. Respecto a las capas ocultas y la cantidad de neuronas por capa, es un parámetro que se deberá estudiar en la construcción del clasificador.

En cuanto al funcionamiento de la red neuronal, se comienza por insertar los valores en las neuronas de la capa de entrada y estos se propagan hacia la siguiente capa aplicando una matriz de pesos *W*, añadiendo un vector llamado *bias* y aplicando una función de activación como puede ser la sigmoide. Este procedimiento se conoce como *forward propagation* [17].

A su vez, estos parámetros utilizados para calcular la salida partiendo de cierta entrada, son modificados mediante el algoritmo de *backpropagation* [17]. Sin entrar en detalle sobre los cálculos matemáticos, cabe mencionar que dicho algoritmo hace uso del error cometido entre la salida de la red y la salida esperada para recalcular las matrices de pesos *W* y los vectores *bias* para cada capa de la red.

2.7. Doc2Vec

Doc2Vec [18] surge como una continuación del modelo word2vec [19], modelo que pretende representar las palabras mediante vectores capaces de capturar la relación existente entre palabras, de modo que la palabra “naranja” sea más similar a “comida” que a “herramienta”. El objetivo es poder realizar operaciones con vectores para obtener un nuevo vector que tenga sentido. El ejemplo más típico utilizado para este caso es el de obtención de la palabra “reina” partiendo del vector “rey" menos el vector “hombre” más el vector “mujer”.

Otra forma de verlo se muestra en la figura 13, donde se puede ver claramente cómo existe la misma relación de “hombre” a “rey” que de “mujer” a “reina”.

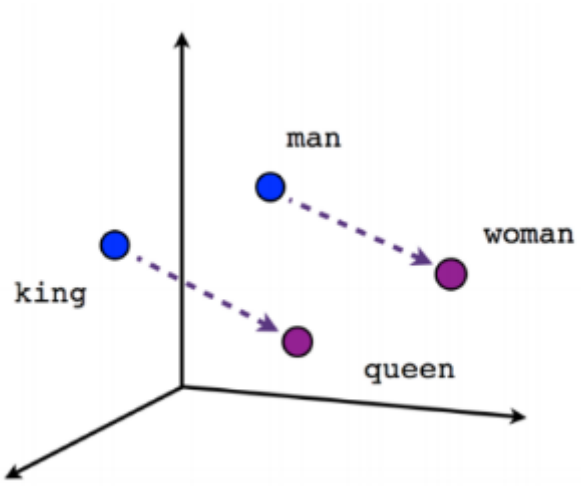

Figura 13: *Ejemplo word2vec*

Word2vec se basa en dos algoritmos, Continuous Bag-of Words (CBOW) [20] y Skip-Gram [20], que se explicarán a continuación de manera sencilla sin recurrir a la implementación exacta.

2.7.1. CBOW

Este algoritmo pretende predecir una palabra dado un contexto. Para ello se utilizará como *input* las palabras del contexto representadas mediante vectores de características y se obtendrá un nuevo vector que representa la palabra que se quería predecir. Tras el entrenamiento, aquellas palabras que aparecen en el mismo contexto tienden a tener una representación en vectores más cercanos. En la figura 14 se ve como se utilizan diferentes palabras para que el clasificador prediga la palabra.

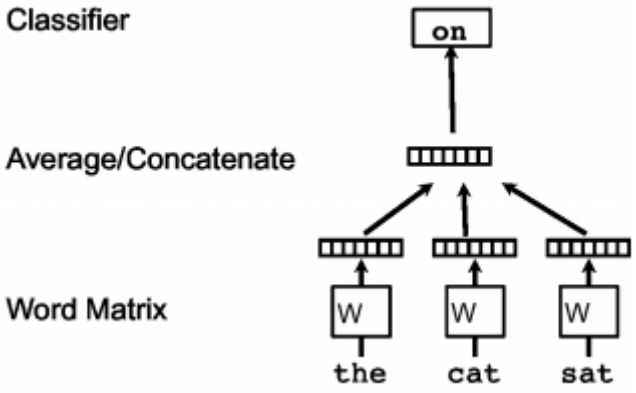

Figura 14: *Estructura CBOW*

2.7.2. Skip-Gram

Este algoritmo pretende predecir el contexto de una palabra de un documento utilizando una red neuronal cuya entrada consiste en un vector de n posiciones, siendo n el tamaño del diccionario, y donde para un cierto caso con un vector con la posición i puesta a uno y el resto de elementos puestos a cero, se pretende obtener como resultado de la red un vector con las mismas n dimensiones donde estarán puestos a uno las posiciones que representen palabras que se encontraban en el contexto de la palabra de la entrada de la red.

Para entender mejor esta idea se explicará un ejemplo mostrado en la figura 15.

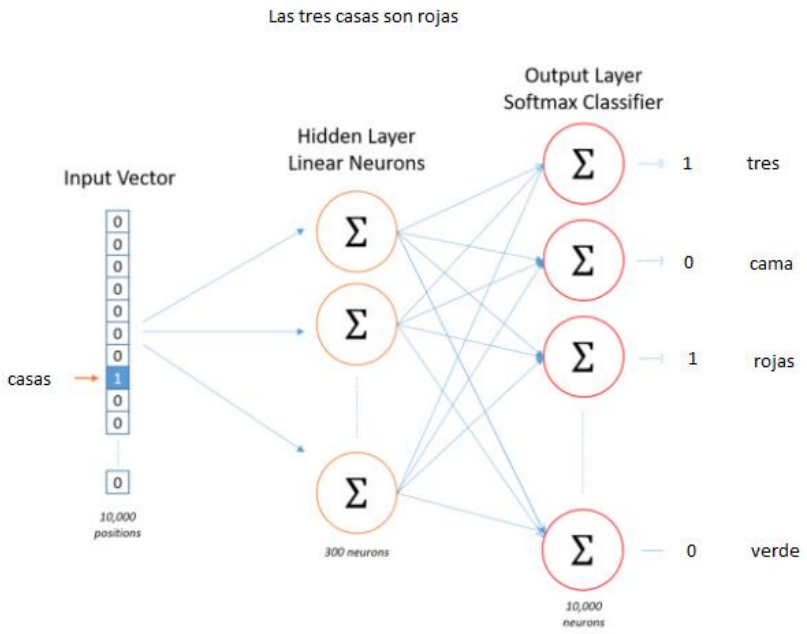

Figura 15: *Estructura Skip-Gram*

Como se puede apreciar en la figura 15, se parte de un documento (“Las tres casas son rojas”) y se pretende predecir palabras como “rojas” y “tres” dada la palabra “casas”, ya que dichas palabras se encuentran en el documento. La red neuronal lo que hará será devolver un vector con un valor cercano al uno en aquellas palabras que predice que están en el contexto de “casas”. Como se puede observar en el ejemplo, la red predice correctamente las palabras adecuadas y deja con un valor cero aquellas palabras que, efectivamente, no se encontraban en el contexto de la palabra de entrada.

Tras haber explicado el funcionamiento de word2vec a grandes rasgos, se puede comentar que Doc2Vec utiliza una idea similar con la pequeña diferencia de que utiliza el documento (es decir, un identificador del mismo) en el entrenamiento de las predicciones, como se muestra en la figura 16. No es objetivo el entendimiento del completo funcionamiento de este modelo, pero sí la idea subyacente utilizada.

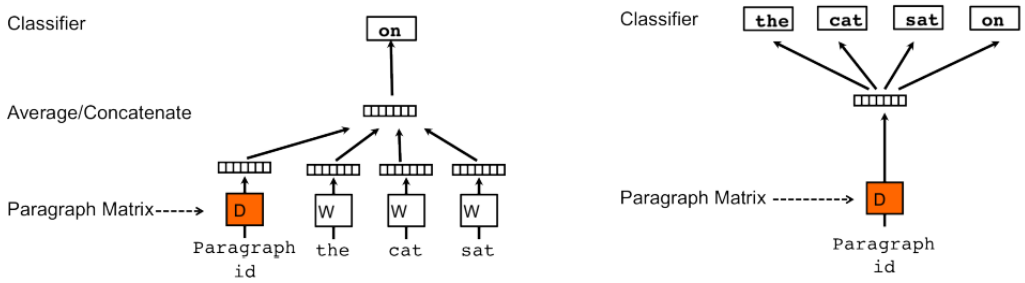

Figura 16: *Estructura Doc2Vec*

3. Tecnologías utilizadas

Este apartado tratará de documentar las tecnologías empleadas en la realización del proyecto, explicando el porqué de su uso.

3.1. Python

Se ha empleado Python como lenguaje de programación dado que dispone de las mejores herramientas para el procesado de lenguaje natural, con librerías como NLTK [21] y spaCy [22], así como scikit-learn [23] en cuanto al *machine learning* se refiere.

3.2. Librerías

En cuanto a las librerías utilizadas, se ha empleado la librería de Python urllib [24] para abrir URLs; Beautiful Soup [25] como *parser* al realizar *web scraping*; NLTK como herramienta que lidera el procesado de lenguaje natural debido a su gran número de librerías y documentación disponible; scikit-learn en cuanto a la construcción de los distintos clasificadores, así como técnicas de reducción de dimensionalidad y extracción de características del texto procesado, ya que dispone de una buena documentación que permite su uso de manera sencilla y está preparada para trabajar junto a librerías como NumPy [26] y SciPy [27], librerías científicas comúnmente utilizadas al trabajar con matrices.

Por otro lado, se encuentra la librería gensim [28] que dispone del modelo Doc2vec y que dispone de todo lo necesario para construir el clasificador correspondiente.

Por último, la librería de python pickle [29] que permite el guardado del clasificador ya entrenado para posteriormente recuperarlo y realizar predicciones.

4. Desarrollo de la solución

En este apartado se detalla el procedimiento seguido a lo largo del proyecto, desde la obtención de los datos utilizados, hasta los resultados obtenidos. En la figura 17 se aprecia el orden seguido.

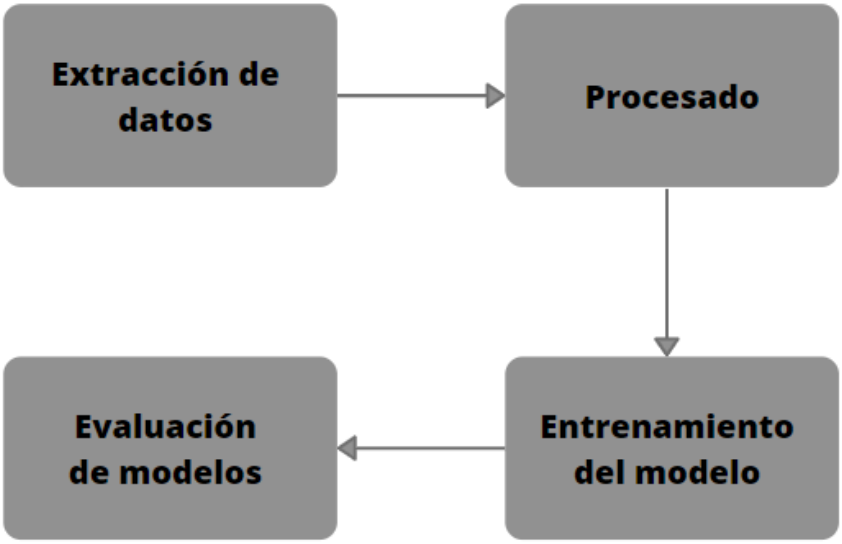

Figura 17: *Esquema del desarrollo del problema*

El orden seguido comienza por la extracción de los titulares tanto satíricos como no satíricos de diferentes lugares web, seguido por el procesado de los datos obtenidos (esto incluye la tokenización del titular, eliminación de *stopwords* y aplicación de *stemming*, para finalmente, utilizar la representación BOW[2]) , entrenamiento de los diferentes modelos utilizados junto a la estimación de los hiperparámetros correspondientes y finalmente la evaluación de los resultados obtenidos para dichos modelos.

[2] Abreviatura para la representación Bag Of Words

4.1. Extracción de datos

El primer paso de este proyecto consistirá en adquirir titulares de noticias, sabiendo de antemano si estas son satíricas o no. Para ello se recurrirá a periódicos que proclaman ser satíricos, así como aquellos que informan de forma seria.

Una vez seleccionada la web deseada, se procederá a realizar *web scraping*, para recuperar solamente los titulares de la página. Inspeccionando el html con el fin de encontrar aquellos elementos que sean titulares, se puede ver que dichos elementos pertenecen a una misma clase, como se puede observar en la figura 18.

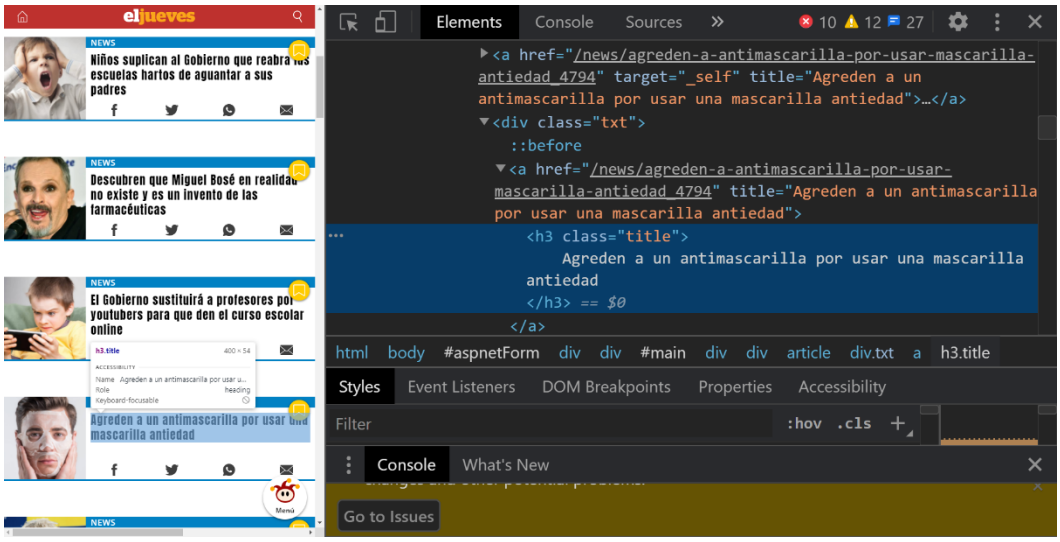

Figura 18: *Html de página web*

Con el objetivo de recuperar dicha información, se comenzará por la obtención del html utilizando urllib, tras lo cual BeautifulSoup será de utilidad para la extracción de información de dicho html. Como resultado de la extracción de los datos, se dispondrá de una lista de titulares de noticias.

Sin embargo, existen ciertos patrones a la hora de redactar los titulares que son propios de cada periódico y no dependen de si son sátira o no. Dichos patrones serán aprendidos por el clasificador a la hora de clasificar, dando resultados mejores de los que debería, por lo cual se deben eliminar dichos titulares con el objetivo de clasificar un título solamente por la información que proporciona y no porque sigue un patrón propio del periódico. Además, se extraerán la misma o parecida cantidad de noticias de cada clase con el propósito de que la probabilidad a priori no afecte a la toma de decisión del clasificador.

De este modo se trabaja con una situación desfavorable pero más justa, donde no se tiene información previa para la clasificación y no existen patrones aparentemente definidos, obteniendo así algo más de veinte mil titulares en total, obtenidos de periódicos como “El País”, “El Mundo”, “El Mundo Today” y “El Jueves”.

4.2. Procesado

Una vez ya se dispone de todos los titulares, marcados como sátira o no sátira en función del periódico de donde provengan, se deberá depurar la información que contienen. A menudo, los documentos (titulares en este caso) se separan en palabras que se incluyen en el diccionario de la represantacion *bag of words*; este proceso se conoce como tokenización, donde cada token es una palabra. Sin embargo, no siempre tiene que ser así: un token puede ser una palabra, un conjunto de dos o más palabras o incluso el lexema de la palabra. En este caso, el proceso de tokenización se ha llevado a cabo mediante el uso del método *word_tokenize* existente en la librería NLTK. Tras la obtención de los tokens, se han eliminado aquellos considerados como *stopwords* y se les ha aplicado *stemming*.

A fin de visualizar mejor la efectividad del preproceso utilizado, se hace una comparativa, mostrada en la tabla 1, donde se puede observar como la talla del diccionario, o lo que es lo mismo, el número de dimensiones de una muestra, disminuye considerablemente al aplicar cada una de las técnicas mencionadas.

Tabla 1: *Talla diccionario tras el preprocesado*

Preprocesado	Talla diccionario
word_tokenize	25.622
word_tokenize+stopwords	25.416
word_tokenize+stopwords+stemming	14.559

Haciendo uso del tokenizador se obtienen 25.622 tokens diferentes de un total de 256.795 palabras totales presentes en el conjunto de titulares, y eliminando *stopwords* se reduce la cantidad de tokens diferentes a 25.416. Aparentemente se ha reducido poco la información eliminada; sin embargo, si se cuenta la cantidad de *stopwords* del total de tokens, se observa cómo el 40,96% (105.196) de los tokens eran considerados *stopwords*. Añadiendo la aplicación de *stemming*, se ha conseguido reducir la dimensionalidad de las muestras casi a la mitad, manteniendo prácticamente toda la información relevante. Esta disminución de dimensionalidad afectará positivamente tanto en el tiempo necesario para entrenar el clasificador como en el tiempo que tarda el clasificador en predecir la clase de una muestra.

Para crear la representación *bag of words* se ha utilizado la librería *CountVectorizer*, perteneciente a scikit-learn, ya que devuelve dicha representación en el formato matriz necesario para el posterior entrenamiento de los modelos. Dicha matriz contiene una gran cantidad de ceros, lo que se conoce como matriz dispersa. Por ello, se almacenará utilizando el formato CSR (Compressed Sparse Row), que, sin entrar en detalles específicos de su implementación, se debe mencionar que reduce de manera considerable la cantidad de memoria utilizada.

Por separado, Doc2Vec utilizará la representación de los títulos ya tokenizados para vectorizar internamente las muestras. Mediante los métodos que ofrece, se entrenará dicho modelo con la mitad de los datos (10.000 muestras aproximadamente), para después utilizar el modelo para vectorizar el resto de las muestras y poder utilizar esa representación diferente de la de *bag of words* para construir un clasificador.

Sin embargo, se deberán estudiar los diferentes parámetros que utiliza el modelo Doc2Vec con el fin de maximizar la precision del clasificador

4.3. Entrenamiento y prueba de modelos

A continuación, se hará uso de diferentes clasificadores ya implementados con Sklearn (scikit-learn), estudiando cómo afectan los diferentes parámetros de cada clasificador y haciendo una comparativa de los resultados obtenidos.

Se comenzará por separar los datos obtenidos en dos subconjuntos, uno de entrenamiento que representará aproximadamente el 80% de los datos totales y otro de test que representará el 20% restante, de modo que el clasificador responderá a muestras nunca antes vistas tras el entrenamiento para la estimación de su precisión. Esto es posible gracias al método *train_test_split* de Sklearn. Como resultado se obtienen cuatro elementos: los datos de entrenamiento, los de test y las etiquetas o clases a las que pertenecen las muestras para cada conjunto de datos. En la figura 19 se muestra el método utilizado, así como el resultado de este.

```
""" TRAIN / TEST SPLIT """
X_train, X_test, y_train, y_test = train_test_split(matrix, matrix_labels, test_size=0.2)
```

Figura 19: *Separación de datos en training y test*

Se puede observar cómo `X_train` representa el conjunto de datos destinado al entrenamiento, `X_test` es el conjunto de datos de test, `y_train` son las etiquetas de clase de `X_train` y `y_test` son las etiquetas de `X_test`. Dicha separación viene dada por el

parámetro `test_size`, cuyo valor varía entre cero y uno e indica el porcentaje de datos destinados a test (0.2 indica que el 20% serán de test y el 80% de *training*).

Tras la separación de los datos, se decide utilizar PCA para la reducción de la alta dimensionalidad de las muestras (alrededor de 15.000 dimensiones) antes de entrenar los modelos. Esto tiende a ser eficaz en las representaciones de imágenes mediante píxeles, dado que algunos apenas aportan información a la hora de realizar la clasificación, aumentando así la dimensionalidad de las muestras con información irrelevante. Sin embargo, pese a ya haber realizado el procesado de los datos para reducir dicha dimensionalidad, se probará cómo afecta PCA a la precisión del clasificador en función de las dimensiones a las que se reduzcan los datos.

Para aplicar PCA deberá elegirse el clasificador a utilizar y, por ello, se estudiarán los diferentes modelos utilizados sin reducción de dimensionalidad, para posteriormente escoger aquel que proporciona mejores resultados y aplicar PCA si dicho clasificador lo permite[3].

Antes de comenzar con el entrenamiento de los clasificadores, cabe mencionar que estos puedes tener hiperparámetros como se ha explicado anteriormente, y si se quiere estudiar cómo estos afectan a la precisión del clasificador y cuál es el óptimo a utilizar en cada caso, se deberán hacer múltiples pruebas para cada combinación posible de los parámetros. Esta tarea puede ser tratada mediante *crossvalidation* haciendo uso de la librería GridSearch, que permite el aplicar dicha técnica para la obtención de los parámetros que maximizan la precisión del clasificador.

El primer caso será el Naive Bayes Multinomial, modelo que utiliza un hiperparámetro `alpha` para el suavizado. Sklearn proporciona la librería MultinomialNB para la creación de dicho clasificador y permite el ajuste de `alpha` pasándolo como parámetro. En la figura 20 se muestran los valores utilizados de `alpha`.

```
param_grid=[
    {"alpha": [1.0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]}
]
```

Figura 20: *Hiperparámetros Naive Bayes Multinomial*

En segundo lugar, se utilizará máquinas de vector soporte para construir el clasificador. Sklearn proporciona la clase SVC (Support Vector Classification) para dicho clasificador y permite el ajuste de hiperparámetros como el kernel, `gamma`, número de iteraciones (por defecto no hay límite), `degree` (grado del kernel polinomial). En la figura 21 se muestran los valores estudiados en cada caso.

[3] Modelos como Naive Bayes Multinomial no permite el uso de matrices con números negativos y en consecuencia PCA no es aplicable

```
param_grid=[
    {"C": [1, 10, 100, 1000],
    "gamma": [1, 0.1, 0.01, 0.001, 0.0001],
    "kernel": ["poly", "rbf"],
    "degree": [2, 3, 4]}
]
```

Figura 21: *Hiperparámetros SVC*

Por ultimo, se utilizará la librería MLPClassifier de Sklearn para la construcción de la red neuronal, estudiando hiperparámetros como la configuración de las capas ocultas (representada como una tupla donde cada elemento de la tupla representa una nueva capa oculta y el valor representa el número de neuronas en dicha capa) y la función de activación entre otros. Se detallan los hiperparámetros estudiados junto a sus valores en la figura 22.

```
param_grid = [
    {"hidden_layer_sizes": [(20,20), (20)],
    "activation": ["tanh", "relu"],
    "solver": ["sgd", "adam"],
    "alpha": [0.0001, 0.05],
    "learning_rate": ["constant","adaptive"],
    "max_iter": [500, 1000, 5000]}
    ]
```

Figura 22: *Hiperparámetros MLP*

Paralelamente, se deberá estimar los parámetros del modelo Doc2Vec para después utilizar algún modelo como el SVC para realizar la clasificación.

Los parámetros a estudiar consisten en el tamaño de ventana (`window`), es decir, el número de palabras adyacentes a considerar como contexto, el número de dimensiones utilizadas para representar una palabra (`vector_size`) y el número de iteraciones (`epochs`) sobre el corpus. Para estudiar la precisión con esta representación de las muestras se ha utilizado el clasificador SVC con los hiperparámetros óptimos, que se detallarán más adelante y resultan ser: `C`=100, `kernel`=rbf y `gamma`=0,001.

El primer parámetro a estimar será el tamaño del vector y para ello se utilizará `window`=2 y `epochs`=25.

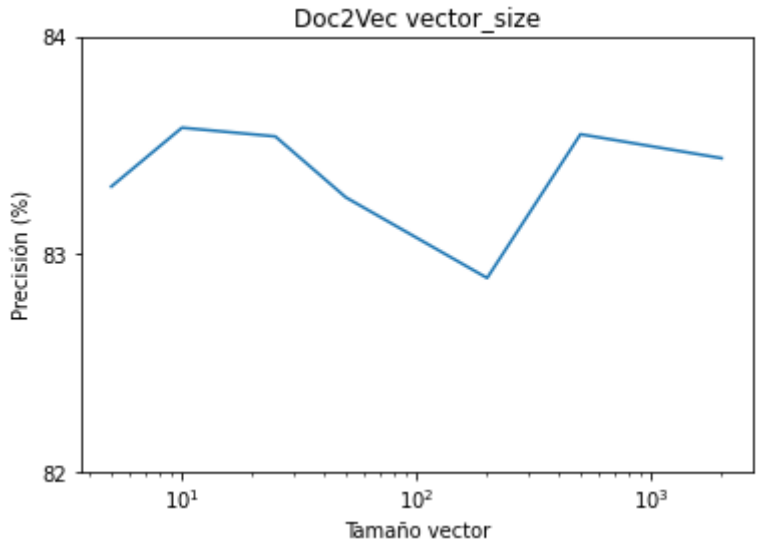

Figura 23: *Doc2Vec, estudio del tamaño del vector*

Como muestra la figura 23, el tamaño del vector no supone una gran influencia en la precisión del clasificador; por ello, y para mantener un número reducido de dimensiones que faciliten el entrenamiento del clasificador, se estudiará el tamaño de la ventana con diez dimensiones manteniendo el valor `epoch` anterior.

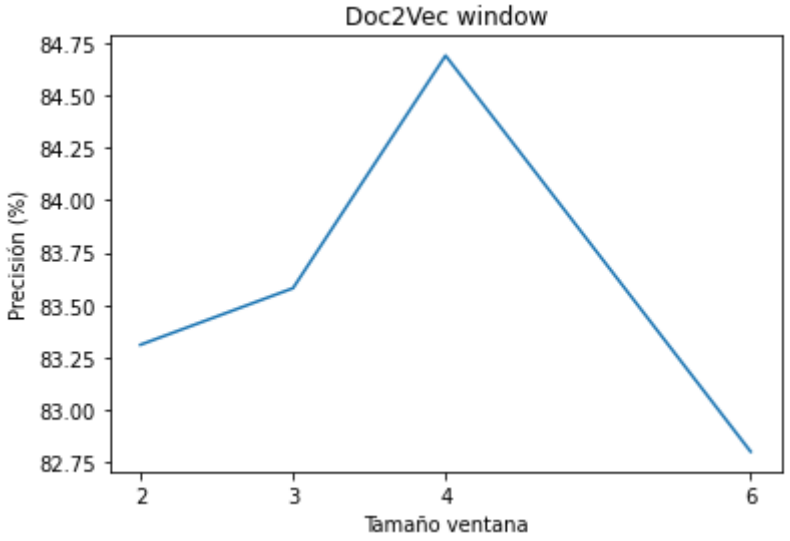

Figura 24: *Doc2Vec, estudio del tamaño de la ventana*

En la figura 24 se muestra cómo el tamaño de la ventana no tiene una gran importancia; sin embargo, se utilizará un tamaño de ventana de cuatro ya que ha dado los mejores resultados de precisión. Juntando las cinco dimensiones del vector y el nuevo tamaño de ventana, se estudiará el efecto del valor de `epoch` sobre la precisión.

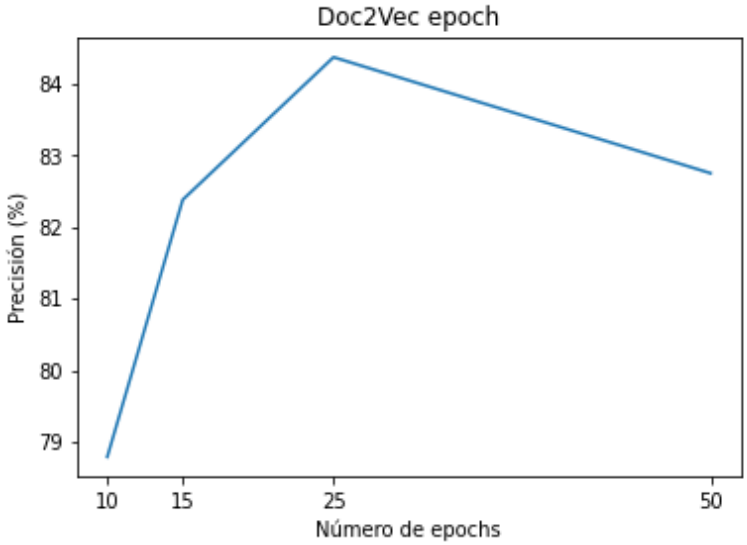

Figura 25: *Doc2Vec, estudio del número de epochs*

Se puede apreciar cómo en la figura 25 el número de *epochs* sí resulta relevante, haciendo que la precisión sea más baja cuando el número de *epochs* es más bajo dado que no se entrena lo suficiente el modelo y más alta cuando las *epochs* son alrededor de 25, siendo este un máximo en la gráfica, ya que la precisión tiende a disminuir cuando las *epochs* toman un mayor valor.

Como resultado se obtiene que la mejor configuración resulta ser `vector_size`=10, `window`=4 y `epoch`=25.

Una vez visto los parámetros a estudiar en cada caso, se deberá entrenar cada clasificador mediante el método *fit*, pasando como parámetros los datos de entrenamiento y las etiquetas de clase correspondientes. Posteriormente, se deberán obtener las predicciones de clase para los datos de test y compararlas con las etiquetas de clase originales para calcular la precisión. Esto se realiza en un solo paso gracias al método *score* que ofrecen las librerías y que realiza dicha operación de predicción y comparación internamente. En la tabla 2 se muestran los resultados obtenidos para cada caso.

Tabla 2: *Precisión clasificadores*

Clasificador	Precisión (%)
Naive Bayes Multinomial	90,80
SVC	90,71
MLP	90,18
Doc2Vec+SVC	84,78

Como se aprecia en la imagen, los resultados obtenidos son similares, obteniendo precisiones en torno al 91% pero algo peor para el modelo Doc2Vec+SVC, consiguiendo una precisión cercana al 85% y siendo el mejor resultado el clasificador naive bayes multinomial con una precisión del 90,80%. A su vez cabe mencionar que los parámetros utilizados para cada clasificador se muestran en la figura 26 y representan los valores de los hiperparámetros óptimos en cada caso.

```
Multinomial: {'alpha': 0.01}

      SVC:  {'C': 100, 'degree': 2, 'gamma': 0.001, 'kernel': 'rbf'}

      MLP:  {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': 20,
             'learning_rate': 'constant', 'max_iter': 500, 'solver': 'sgd'}
```

Figura 26: *Hiperparámetros óptimos*

Sin embargo, si se repitiese el proceso, esta precisión sería diferente, pero con un valor similar. Esto se debe a la aleatoriedad a la hora de seleccionar los datos de *training* y test. A

su vez, es lógico pensar que si se tienen pocas muestras de test, el valor de la precisión no aporta mucha confianza; esto se ve claro con un ejemplo extremo donde tengamos una sola muestra para el test y resulta que el clasificador la clasifica correctamente; sin embargo, pese a que la precisión es del 100%, el haber utilizado una sola muestra hace que no se tenga mucha confianza en cómo responderá cuando se le proporcione otra muestra, por ello, se debe calcular el intervalo del 95% de confianza para tener una medida de cómo de seguro se está respecto a dicha precisión. Esto se calcula mediante la fórmula expuesta a continuación.

$$P(\hat{p} - \epsilon \leq p \leq \hat{p} + \epsilon) = 0{,}95; \qquad \epsilon = 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}$$

Siendo p̂ el número de errores cometidos entre el número total de muestras, N como el número total de muestras y ϵ el error para dicho intervalo al 95%.

En la tabla 3 se expresa dicho error para cada clasificador disponible.

Tabla 3: *Error intervalo confianza 95%*

Clasificador	Error (%)	Precisión ± Error (%)
Naive Bayes Multinomial	0,854	89,94 – 91,65
SVC	0,858	89,85 – 91,56
MLP	0,879	89,30 – 91,05
Doc2Vec+SVC	1,061	83,71 – 85,84

Como se puede apreciar en la tabla 3, en la tercera columna se calcula dicho intervalo de confianza, el cual indica que tanto el clasificador naive bayes multinomial como SVC y MLP tienen una precisión similar y sus diferencias no resultan significativas. Caso completamente diferente es el clasificador Doc2Vec+SVC, cuyo intervalo no se solapa con el resto de los intervalos y cuya precisión resulta ser algo inferior.

Una vez estudiada la precisión de los distintos clasificadores, se puede optar por estudiar cómo afectan ciertos procedimientos a la hora de la clasificación. Entre estos se pueden encontrar la aplicación de PCA, con el objetivo de reducir la dimensionalidad y con ello el tiempo invertido en el entrenamiento de los clasificadores, así como el tiempo que se tarda en, dada una nueva muestra, vectorizarla y predecir la clase a la que pertenece.

Se hará uso del clasificador SVC con los hiperparámetros óptimos, dado que obtiene una buena precisión, similar a la mejor precisión dada por el clasificador naive bayes multinomial.

Para hacerse una idea de cómo de buena es esta técnica, suele ser conveniente el estudio del *scree plot*, una gráfica que muestra la proporción de varianza explicada por las n componentes principales. En otras palabras, indica la información retenida en las n dimensiones a las que se va a transformar, de modo que será mejor dicha representación en n dimensiones cuanto más cercano al 100% sea la suma de la proporción de varianza explicada para las n dimensiones. En la figura 27 se muestra dicha gráfica dado n = 100.

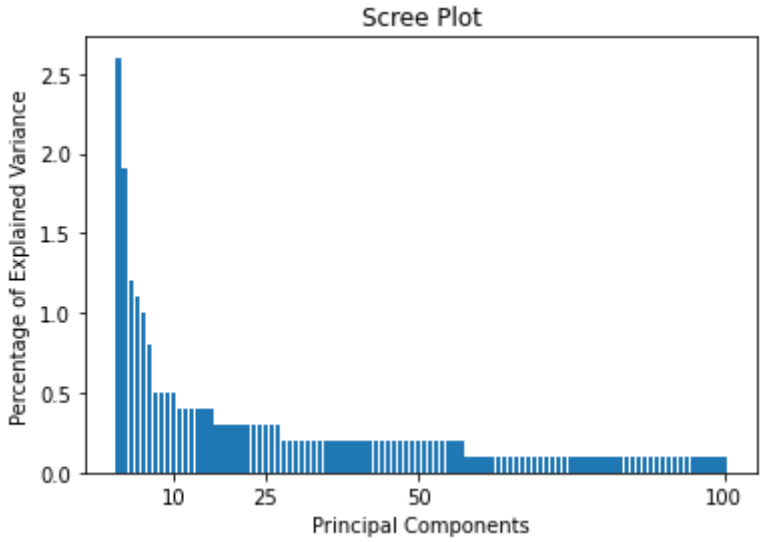

Figura 27: *Scree plot*

La gráfica muestra como la primera componente representa un 2,5% aproximadamente del total de varianza, lo que supone un valor considerablemente bajo, y este valor va disminuyendo de manera considerable para las siguientes componentes. A la hora de seleccionar el número de dimensiones a las que se realizará la transformación, se tendrá en cuenta la gráfica, y viendo a partir de cuántas componentes el porcentaje de varianza explicada disminuye de manera considerable. Para este caso se puede observar como a partir de las 60 dimensiones, la varianza explicada comienza a ser poco significativa.

Otra forma de seleccionar el número de componentes es representar el porcentaje de varianza explicada acumulada; esto queda representado en la figura 28.

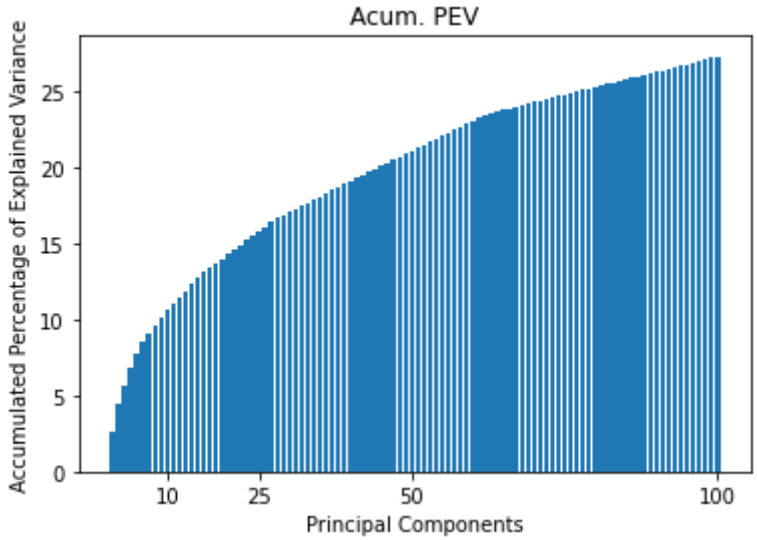

Figura 28: *Acum. PEV plot*

Como se observa, la ganancia de porcentaje de varianza acumulada es sustancial para aproximadamente las primeras cincuenta dimensiones, lo que sugiere que esas sean las dimensiones al aplicar la transformación. Sin embargo, con dichas dimensiones apenas se consigue un 20% de la varianza explicada, lo que supone una cifra muy baja como para representar los datos con dichas dimensiones.

Teniendo esto en cuenta, se procede a estudiar cómo la dimensionalidad utilizada afecta a la precisión del clasificador utilizado (SVC) con los hiperparámetros óptimos.

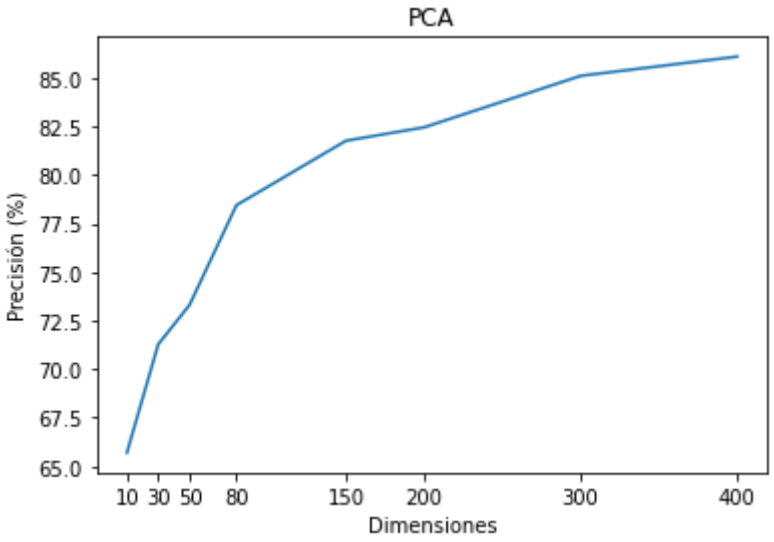

Figura 29: *PCA dimensiones*

Como se aprecia en la figura 29, a medida que se aumentan las dimensiones aumenta la precisión del clasificador, pero a partir de las 150 dimensiones el porcentaje de precisión ganado tiende a disminuir de manera considerable. Pese a todo, la precisión sigue siendo menor que la obtenida sin aplicar PCA.

Como se ha visto, PCA no aporta una mejora, por lo que es más conveniente no utilizarlo y mantener la precisión obtenida anteriormente.

Otra opción a estudiar es el cómo afecta el uso de diferentes representaciones de las muestras. Hasta ahora, se han representado los títulos como tokens compuestos por una palabra que no forme parte de la lista de *stopwords* a la que se le ha aplicado *stemming*. Otra representación común es el uso de n-gramas, donde un token ya no es una sola palabra, sino que pasa a ser un conjunto de n palabras. Para comprobar si esta representación resulta más efectiva a la hora de clasificar, se estudiará el uso de bigramas y trigramas.

Gracias a Sklearn, este proceso se resuelve de manera sencilla utilizando la librería anteriormente mencionada CountVectorizer. Esta proporciona un parámetro a la hora de construir la representación *bag of words* que permite hacer uso de unigramas, bigramas, combinar unigramas y bigramas, etc. En este caso se estudiará haciendo uso de unigramas, bigramas junto a unigramas, bigramas solamente, trigramas junto a bigramas y unigramas y trigramas solamente. Esto viene representado en la tabla 4 por el índice i que representa el i-grama mínimo considerado y el índice j que representa el j-grama máximo. Un ejemplo es el caso i=1 y j=3, representando que se considerarán desde unigramas, pasando por bigramas, hasta trigramas. Esto supone un gran aumento de la dimensionalidad de los datos, valor que debe tenerse en cuenta a la hora de utilizar una representación de los datos u otra.

Cabe mencionar que el clasificador utilizado es el naive bayes multinomial junto a los hiperparámetros óptimos dado que obtiene la mejor precisión conseguida hasta el momento.

Tabla 4: *Representación n-gramas*

Índice i	Índice j	Precisión (%)	Dimensionalidad
1	1	90,80	14.559
1	2	91,22	115.394
2	2	80,45	100.835
1	3	90,20	219.992
3	3	57,05	104.598

Se observa en la tabla cómo el aumentar los n-gramas considerados no siempre aumenta la precisión del clasificador; utilizar una representación de unigramas o unigramas junto a bigramas obtiene un mejor resultado que utilizar solamente bigramas. Parece ser que el uso de unigramas, unigramas junto a bigramas o unigramas, bigramas y trigramas obtienen un porcentaje de precisión en torno al 91%, mientras que representarlo por bigramas solamente pierde hasta un 10% de precisión, y peor es el caso donde solo se consideran trigramas, obteniendo una precisión del 57%.

Por último, se estudiará cómo el volumen de datos utilizado para el entrenamiento de los diferentes modelos afecta la precisión, con el objetivo de conocer si el aumento de la cantidad de muestras supone un factor a considerar para la mejora del clasificador. Sin embargo, aumentar el número de muestras más allá de las 21.000 manteniendo un número semejante de muestras satíricas y no satíricas resulta complicado, dado que requeriría aumentar la base de datos de la que se dispone. En la tabla 5 se refleja cómo el volumen de datos influye en la precisión del clasificador multinomial haciendo uso de unigramas y bigramas.

Tabla 5: *Estudio del volumen de datos*

Número de muestras	Precisión (%)
≈ 5000	91,70
≈ 10000	91,51
≈ 20000	91,22

Observando los datos obtenidos, se aprecia como prácticamente la precisión es la misma para los tres casos. Esto indica que existen muestras suficientes para entrenar el clasificador y la precisión no mejora al aumentar el tamaño de las muestras utilizadas.

Una vez visto cómo afecta cada parámetro por separado, se pretende combinar la mejor solución para cada caso con el objetivo de construir el mejor clasificador y estudiar lo que se

conoce como matriz de confusión, matriz que indica el comportamiento del clasificador para cada clase. Esto permite visualizar si el clasificador obtiene una precisión similar para ambas clases o, por el contrario, tiende a clasificar una clase mejor que la otra. En la figura 30 se representa dicha matriz utilizando un clasificador multinomial utilizando unigramas y bigramas.

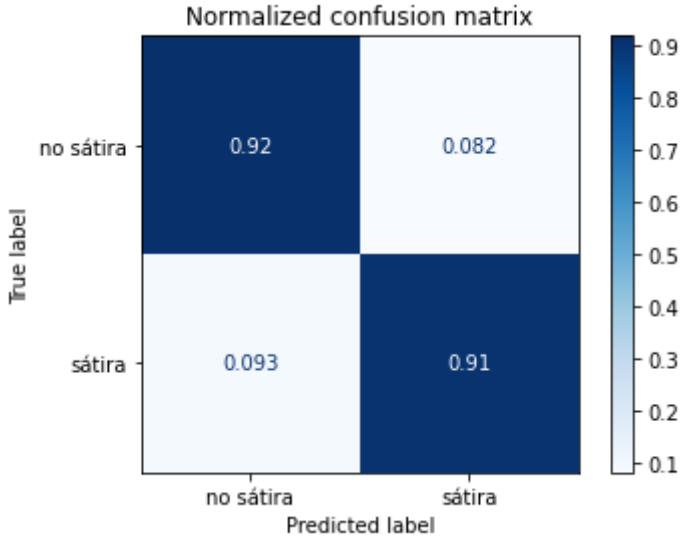

Figura 30: *Matriz de confusión*

Como se puede observar en la figura 30, donde se ha normalizado la matriz con el objetivo de tener una mejor visión de la precisión, ambas clases tienden a clasificarse bien con una precisión aproximada del 91%. Esto puede llegar a ser relevante especialmente cuando se trata con múltiples clases, dado que puede darse el caso en el que se clasifique perfectamente unas clases pero otras no sepa clasificarlas.

Tras un estudio detallado a fin de construir el mejor clasificador posible, se guardará dicho clasificador en un archivo como un objeto. Para ello se utilizará la librería pickle, permitiendo guardar el objeto y recuperarlo en un futuro. En caso de que se requiera el uso del clasificador, bastará con extraerlo del fichero en el que se encuentra y aplicar el método deseado de la librería Sklearn correspondiente (en este caso el clasificador es naive bayes multinomial, perteneciente a la clase MultinomialNB [30]). Este proceso de guardado y carga del objeto se muestra en la figura 31.

```
# guardar clasificador
with open('MiClasificador.pkl', 'wb') as f:
    pickle.dump(clasificador, f)

# cargar clasificador
with open('MiClasificador.pkl', 'rb') as f:
    clf = pickle.load(f)
```

Figura 31: *Guardar y cargar clasificador*

Posteriormente al guardado, se pretende recuperar dicho clasificador para ver su comportamiento en acción. En concreto, se pretende hacer uso de algunas muestras seleccionadas manualmente de diferentes páginas web para observar cómo el clasificador predice una muestra y la seguridad de dicha predicción. La tabla 6 muestra dichos ejemplos junto a la clase que verdaderamente pertenece y la probabilidad de pertenencia que el clasificador estima para cada clase.

Tabla 6: *Ejemplos de clasificación*

Titular de noticia	Clase a la que pertenece	Predicción	
		Probabilidad de sátira	Probabilidad de no sátira
El Kia Sorento, elegido coche del año en el que orinar por la Asociación Europea de Perros	Sátira	0,997	0,003
Despiden a la empleada de un hotel de EEUU tras llamar a la Policía porque una familia afroamericana estaba usando la piscina	No sátira	0,065	0,935
Piden la retirada de la esclavitud por ensalzar el racismo	Sátira	0,016	0,984
España enviará 100.000 litros de sangría y 1.000 balcones a Reino Unido como ayuda para pasar el verano	Sátira	0,565	0,435

Con esto se pretende mostrar como el clasificador predice los ejemplos mostrados. Para el primer ejemplo, el clasificador proporciona una seguridad del 99,7% de que es sátira, y efectivamente lo es. Esto puede deberse a que temas que hablen de perros orinando en coches no es algo muy común en noticias serias y es por eso que el clasificador tiene una gran confianza en su decisión. El segundo caso puede aparentar irónico y tratarse de una broma, sin embargo no lo es, y el clasificador, pese a disminuir su seguridad de decision a un 93,5%, acierta. Algo muy diferente ocurre con la tercera muestra, donde a pesar de ser obvia la sátira del titular, el clasificador la clasifica erróneamente pese a estar más de un 98% seguro de que no es sátira. Por último, destacar que existen casos donde el clasificador no tiene una gran seguridad a la hora de tomar la decisión. Este es el caso de la cuarta muestra, clasificada correctamente pero sin tener una gran seguridad (56,5%).

5. Conclusión

Como conclusión, se ha conseguido construir un clasificador de texto capaz de reconocer titulares de noticias satíricos, alcanzando una precisión algo superior al 91%, una cifra aparentemente bastante satisfactoria.

Se ha cumplido con los objetivos propuestos de la extracción de las muestras automáticamente de diferentes lugares web, tarea que puede llegar a ser algo compleja debido a la falta de bases de datos que recojan los titulares, la diferente disposición de las páginas web oficiales y la escasez de periódicos dedicados a la publicación de noticias satíricas.

A su vez, se ha conseguido hacer la comparativa de precisión en función de los diferentes modelos utilizados, estudiando en cada caso la influencia de los hiperparámetros, estimándolos mediante la técnica de *crossvalidation*.

Por último, se han utilizado diferentes métodos de preprocesado de texto y reducción de dimensionalidad con el fin de ver cómo estos repercuten en el porcentaje de acierto del clasificador, observando que principalmente permiten reducir el número de características de las muestras, ahorrando memoria del sistema y acelerando su computación para el entrenamiento de los modelos, pero sin mejorar la tarea de clasificación en cuanto a precisión se refiere.

Al mismo tiempo, se ha conseguido enfrentarse a un problema real partiendo desde cero, donde lo único que se posee es la experiencia y conocimiento adquirido de haber cursado las asignaturas de la carrera. Esto ha supuesto un refuerzo y asentamiento de las bases teóricas, así como la expansión del conocimiento de las herramientas disponibles en el campo del *machine learning*, desde el uso de herramientas ya conocidas, similares a las disponibles en las librerías BeautifulSoup, NLTK y Sklearn, hasta la introducción de nuevas herramientas como el modelo Doc2Vec.

5.1. Relación del trabajo realizado con las asignaturas cursadas

Este trabajo surge como fruto de haber estudiado asignaturas como “Sistemas Inteligentes”, dando una base de conocimiento en el campo del *machine learning*, base que será reforzada y expandida en asignaturas como “Percepción” y “Aprendizaje Automático”. En particular, la asignatura “Percepción” proporciona el conocimiento necesario para saber abordar este tipo de problemas de clasificación de texto, proponiendo las herramientas y metodología básica utilizada. A su vez, estas herramientas se profundizan en asignaturas como “Sistemas de Almacenamiento y Recuperación de la información”, proporcionando un mayor conocimiento en cómo tratar los datos y extraer la información más relevante. Otras asignaturas como “Aprendizaje Automático”, enseña, entre otras cosas, el uso de los

diferentes modelos utilizados en este trabajo, así como el funcionamiento detallado de cada uno de ellos.

Pese a todo, se debe destacar que cada asignatura contribuye parcialmente a la realización de este trabajo, ya sea en menor o mayor medida y el uso y entendimiento de las herramientas no sería posible sin el conocimiento de otras muchas asignaturas como “Estadística”, “Análisis Avanzado de Datos” y “Computación Científica”.

Las asignaturas previamente mencionadas contribuyen principalmente al conocimiento necesario, pero habilidades como el pensamiento crítico, la habilidad para solucionar problemas antes no vistos, la capacidad de ver las cosas con diferentes perspectivas y el manejo adecuado y justificado de cada herramienta según el problema tratado son, entre muchas otras, habilidades que se adquieren y perfeccionan especialmente en la rama de computación.

6. Trabajo futuro

Para la resolución de problemas de clasificación de texto existen muchos métodos que se pueden emplear. Por ello, en este proyecto se ha considerado utilizar aquellos más comunes y con mejores expectativas; sin embargo, la capacidad de cómputo disponible limita en parte el estudio a la hora de utilizar las diferentes herramientas. Un ejemplo de esto sucede al estimar los hiperparámetros de los clasificadores, tarea costosa como consecuencia de las múltiples combinaciones de parámetros a utilizar.

Además, son múltiples las alternativas disponibles para la construcción del clasificador: pudiendo utilizar modelos disponibles como mixtura de gaussianas, K-medias, K-NN, o árboles de decisión entre otros.

No obstante, un modelo no obtendrá buenos resultados por muy potente que sea si la representación de los datos no es buena. He aquí la esencia del rendimiento del clasificador, una buena representación de las muestras facilita increíblemente la tarea de clasificación. Sin embargo, esto no es tan sencillo cuando se trata con textos; por ello existen algunas técnicas como análisis de sentimientos o lematización (similar al *stemming*), técnicas no estudiadas en este trabajo, que podrían incluirse.

Pese a todo, la sátira es algo subjetivo que no se rige por normas y utilizar solamente texto escrito puede no ser suficiente para identificarla correctamente; es por eso por lo que es necesario probar diferentes herramientas, combinarlas entre sí y analizar su efectividad.

Todo esto nos permite crear el clasificador, el cual puede ser utilizado para construir una aplicación móvil o una extensión en el navegador que nos permita identificar los títulos de una noticia e indique si esta es satírica o no y con qué porcentaje de seguridad lo puede decir. Esto puede resultar especialmente útil para redes sociales, donde la información se extiende rápidamente y a menudo se desinforma a la gente de manera inintencionada, reenviando mensajes y noticias de fuente desconocida, descontextualizando la información y dando lugar a dudas sobre la veracidad de esta.

7. Referencias

[1] M. Bassiouni, M. Ali & E. A. El-Dahshan (2018) Ham and Spam E-Mails Classification Using Machine Learning Techniques, Journal of Applied Security Research, 13:3, 315-331, DOI: 10.1080/19361610.2018.1463136. Disponible en: https://www.tandfonline.com/doi/full/10.1080/19361610.2018.1463136

[2] De la Hoz Manotas, A. K., Martínez-Palacio, U. J. y Mendoza-Palechor, F. E. (2013). Técnicas de ml en medicina cardiovascular. Memorias, 11(20), 41-46. Disponible en: https://www.researchgate.net/profile/Alexis_De_La_Hoz_Manotas/publication/279850557_Tecnicas_de_ml_en_medicina_cardiovascular/links/5806544808ae0075d82c6296/Tecnicas-de-ml-en-medicina-cardiovascular.pdf

[3] Polyakov E.V., Mazhanov M.S., Rolich A.Y., Voskov L.S., Kachalova M.V., Polyakov S.V. Investigation and Development of the Intelligent Voice Assistant for the Internet of Things Using Machine Learning. Disponible en: https://fardapaper.ir/mohavaha/uploads/2019/09/Fardapaper-Investigation-and-development-of-the-intelligent-voice-assistant-for-the-Internet-of-Things-using-machine-learning.pdf

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Demis Hassabis. AlphaZero: Shedding new light on chess, shogi, and Go [en línea]. Diciembre 2018. Disponible en: https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go

[5] Joaquín Amat Rodrigo. Análisis de Componentes Principales (Principal Component Analysis), PCA) y t-SNE [en línea]. Junio 2017. Disponible en: https://www.cienciadedatos.net/documentos/35_principal_component_analysis

[6] Bellman R.E. Adaptive Control Processes. Princeton University Press, Princeton, NJ, 1961. Disponible en: https://books.google.es/books?hl=es&lr=&id=iwbWCgAAQBAJ&oi=fnd&pg=PR9&ots=bDKeTmB35i&sig=4gVnqLbpelMTl44ax9xtII_JJoo&redir_esc=y#v=onepage&q&f=false

[7] Harry Zhang. The Optimality of Naive Bayes [en línea]. Disponible en: https://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf

[8] Ashraf M. Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial Naive Bayes for Text Categorization Revisited [en línea]. 2004. Disponible en: https://perun.pmf.uns.ac.rs/radovanovic/dmsem/cd/install/Weka/doc/pubs/2004/KibriyaAI04-MultinomialNBRevisited.pdf

[9] McCallum, A., Nigam, K.: A Comparison of Event Models for Naive Bayes Text Classification. In: AAAI/ICML-98 Workshop on Learning for Text Categorization. (1998)

41–48. Disponible en: https://www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf

[10] Fernando Pérez Nava. [en línea]. Disponible en: https://fdoperez.webs.ull.es/doc/recpat5.pdf

[11] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. Enero 2008. Disponible en: https://projecteuclid.org/download/pdfview_1/euclid.aos/1211819561

[12] Joaquín Amat Rodrigo. Máquinas de Vector Soporte (Support Vector Machines, SVMs) [en línea] Abril 2017. Disponible en: https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines

[13] Tom Dietterich. Overfitting and Undercomputing in Machine Learning. 1995. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.2069&rep=rep1&type=pdf

[14] Support Vector Machines with the mlr package [en linea]. [actualizado 10 octubre 2019]. Disponible en: https://machinelearningwithmlr.wordpress.com/2019/10/10/example-post/

[15] https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa

[16] Jason Brownlee. 3 Ways to Encode Categorical Variables for Deep Learning [en línea]. Noviembre 2019. Disponible en: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/

[17] Aston Zhang, Zarchary C. Lipton, Mu Lu, Alexander J. Smola. Dive into Deep Learning [en línea]. Agosto 2020. Disponible en: https://d2l.ai/d2l-en.pdf

[18] Quoc Le, Tomas Mikolov. Distributed Representations of Sentences and Documents [en línea]. Mayo 2014. Disponible en: https://arxiv.org/pdf/1405.4053v2.pdf

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality [en línea]. 2013. Disponible en: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[20] Gonzalo Ruiz de Villa. Introducción a Word2vec (skip gram model). [en línea] Mayo 2018. Disponible en: https://medium.com/@gruizdevilla/introducci%C3%B3n-a-word2vec-skip-gram-model-4800f72c871f

[21] Documentación de la librería NLTK. Disponible en: https://www.nltk.org/

[22] Documentación de la librería spacy. Disponible en: https://spacy.io/

[23] Documentación de la librería scikit-learn. Disponible en: https://scikit-learn.org/

[24] Documentación de la librería urllib. Disponible en: https://docs.python.org/3/library/urllib.html

[25] Documentación de la librería BeautifulSoup. Disponible en: https://www.crummy.com/software/BeautifulSoup/

[26] Documentación de la librería numpy. Disponible en: https://numpy.org/

[27] Documentación de la librería scipy. Disponible en: https://www.scipy.org/

[28] Documentación de la librería Doc2Vec. Disponible en: https://radimrehurek.com/gensim/index.html

[29] Documentación de la librería pickle. Disponible en: https://docs.python.org/3/library/pickle.html

[30] Documentación de la clase MultinomialNB. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB