



ESCUELA TÉCNICA SUPERIOR DE INGENIEROS  
INFORMÁTICOS

UNIVERSIDAD POLITÉCNICA DE MADRID

---

# Filtrado inteligente de noticias basado en Deep Learning

---

TRABAJO FIN DE MÁSTER  
MÁSTER UNIVERSITARIO EN INGENIERÍA  
INFORMÁTICA

AUTORA: Ana Isabel Lopera Martínez  
TUTOR: Emilio Serrano Fernández

2017/2018



## Agradecimientos

*Gracias a mi tutor por guiarme en la elaboración de este trabajo.*

*Gracias a mis padres, por enseñarme el valor de no rendirme nunca.*



## Resumen

El análisis de posicionamiento de noticias consiste en identificar la disposición de una noticia con respecto a un hecho o entidad, y supone una tarea fundamental para determinar la veracidad de ésta. Este trabajo presenta una solución sobre el problema presentado en el desafío de Fake News Challenge, proponiendo un servicio web de filtrado de noticias basado en redes neuronales hacia delante. La herramienta está orientada a usuarios finales y les permitirá obtener la posición de una noticia concreta respecto del titular. Para el desarrollo del clasificador se han comparado representaciones basadas en el método de bag of words y vectores de medias de embedding de word2vec. La representación del titular y el cuerpo de la noticia alimenta un modelo de red hacia delante que realiza una clasificación multiclase. El modelo final consiste en una red neuronal optimizada con tres capas ocultas aplicando regularización L2, tomando como entrada la representación de vectores de medias. Se consigue un resultado de 87% en valor predictivo positivo y 86% en términos de recall.



## Summary

Analysis of news stance consists on addressing the position on a report towards a fact or entity, and takes an important role in the assessment of veracity of the report itself. This project proposes a solution relating to the Fake News Challenge problem, and proposes a web service for news filtering, based on feedforward neural networks. This tool is conceived to end-users and will let them obtain the stance of a text with respect a claim. To develop the classifier, text representations based on bag-of-words and vector embedding mean have been considered. Text representations of headline and article body are used as input for a feedforward neural network trained to make a multiclass classification. The final model proposed consists on a hand-tuned 3-layer feedforward network applying L2 regularization, taking as input two vectors of means of word embeddings. This model achieves a 87 % in terms on precision and a 86 % in terms on recall.





# Índice

<b>1. Introducción y objetivos</b>	<b>1</b>
<b>2. Estado del arte</b>	<b>3</b>
2.1. Procesamiento de Lenguaje Natural. Representación de textos . . . . .	3
2.1.1. Bag Of Words (BOW) . . . . .	4
2.1.2. Modelos basados en redes de neuronas . . . . .	5
2.1.3. Embedding vectors. Word2Vec . . . . .	5
2.2. Redes de neuronas artificiales . . . . .	10
2.3. TensorFlow . . . . .	14
2.3.1. Definición de grafos de computación . . . . .	16
2.4. Técnicas de optimización y regularización de Redes neuronales. . . . .	18
2.4.1. Métodos de optimización. . . . .	18
2.4.2. Técnicas de regularización en Redes de Neuronas . . . . .	19
2.4.3. Modelos de clasificación a partir de datasets desbalanceados. Técnicas correctoras . . . . .	22
2.5. Trabajos relacionados . . . . .	24
2.5.1. Detección de posición. Técnicas de clasificación de textos. . . . .	25
2.5.2. Discusión de técnicas y aportación del trabajo. . . . .	32
<b>3. Evaluación de riesgos</b>	<b>35</b>
3.1. Caracterización del problema . . . . .	35
3.2. Dataset para la clasificación de noticias. . . . .	36
3.3. Filtrador web inteligente de noticias. . . . .	37
<b>4. Resultados</b>	<b>41</b>
4.1. Modelos de representación de textos . . . . .	41
4.1.1. Metodología . . . . .	41
4.1.2. Resultados . . . . .	55
4.1.3. Discusión . . . . .	62
4.2. Clasificadores basados en Redes Neuronales Profundas . . . . .	64
4.2.1. Modelo de Red Neuronal Hacia Delante . . . . .	64
4.2.2. Resultados . . . . .	72
4.2.3. Discusión . . . . .	80
4.3. Filtrador Web de Noticias . . . . .	81
4.3.1. Arquitectura del sistema . . . . .	81
4.3.2. Interfaz de acceso a la plataforma. . . . .	84
<b>5. Conclusiones</b>	<b>85</b>
<b>6. Líneas Futuras</b>	<b>87</b>



## Índice de figuras

1.	Ejemplos de analogías que pueden ser obtenidas a partir de un modelo de word embeddings . . . . .	6
2.	Arquitectura de CBOW [35] . . . . .	7
3.	Arquitectura de Skip-gram [35] . . . . .	8
4.	Diagrama sobre la predicción de términos en base al Paragraph Vector y los términos del contexto [31] . . . . .	9
5.	Variante de Bag Of Words distribuida [31] . . . . .	10
6.	Perceptrón Simple (Linear Threshold Unit) . . . . .	11
7.	Perceptrón Multicapa (Multilayer Perceptron, MLP) con una capa oculta . . . . .	12
8.	Pila de APIs ofrecidas por TensorFlow [60] . . . . .	15
9.	Ejemplo de grafo de computación, que aplica el Teorema de Pitágoras	17
10.	Expresiones para el Momentum Clásico [58] . . . . .	19
11.	Expresiones para el Gradiente Acelerado de Nesterov [58] . . . . .	19
12.	Red original (izquierda) y red resultante tras aplicar Dropout (derecha) [40] . . . . .	21
13.	Función de coste aplicando la regularización de Lasso (L1) . . . . .	21
14.	Función de coste aplicando la regularización de Ridge (L2) . . . . .	21
15.	Diagrama lógico del filtrador web de noticias . . . . .	38
16.	Flujo de procesamiento de muestras. . . . .	39
17.	pruebas sobre el modelo oficial de Word2Vec, utilizando el método most similar. . . . .	45
18.	pruebas sobre el modelo propio, utilizando el método most similar . .	47
19.	Grafo de computación del modelo de Perceptrón Multicapa . . . . .	51
20.	Clasificaciones de las particiones de entrenamiento y test (Particionado 1) . . . . .	53
21.	Clasificaciones de las particiones de entrenamiento y test (Particionado 2) . . . . .	54
22.	Matrices de confusión de la configuración final para la representación de Bolsa de palabras, particionado 1 . . . . .	56
23.	Curvas ROC por clase para el modelo de bolsa de palabras, sobre la partición 1. . . . .	56
24.	Matrices de confusión de la configuración por defecto para la representación de Bolsa de palabras, sobre la partición 2. . . . .	58
25.	Curvas ROC por clase para el modelo de bolsa de palabras, sobre la partición 2. . . . .	58
26.	Matrices de confusión sobre el conjunto de test en los modelos de Vectores de Medias . . . . .	60
27.	Curvas ROC del modelo de vector de medias, particionado 1 . . . . .	60
28.	Matrices de confusión sobre el conjunto de test en los modelos de Vectores de Medias, sobre la partición 2 . . . . .	61
29.	Curvas ROC del modelo de vector de medias, particionado 2 . . . . .	62

30.	Función de Accuracy y de coste para la configuración inicial, sobre el conjunto de entrenamiento/validación, particionado 2 . . . . .	66
31.	Patrón de actualización de la tasa de aprendizaje. . . . .	67
32.	Configuraciones de learning rate dinámico utilizadas en la experimentación. . . . .	68
33.	Código de definición de la capa de dropout entre dos capas ocultas . .	70
34.	Arquitectura de la red con las capas de dropout . . . . .	70
35.	Código del cálculo de la función de coste con el factor de penalización	71
36.	Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, sobre el conjunto de entrenamiento/validación. .	74
37.	Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, sobre el conjunto de entrenamiento/validación. .	75
38.	Curvas ROC de la configuración final sobre el conjunto de test . . . .	76
39.	Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación . . . . .	78
40.	Comparación de las diversas configuraciones sobre técnicas de regularización, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación . . . . .	79
41.	Comparación de las diversas configuraciones sobre técnicas de regularización, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación . . . . .	79
42.	Pila tecnológica del clasificador web de noticias. . . . .	82
43.	<i>Stance Detector</i> : Formulario de envío de noticias para su clasificación	84
44.	<i>Stance Detector</i> : captura de visualización de clasificación. . . . .	84

## Índice de cuadros

1.	Resultados de validación con el modelo de bolsa de palabras, particionado 1 . . . . .	55
2.	Resultados de test con el modelo de bolsa de palabras, particionado 1.	55
3.	Resultados de validación con el modelo de bolsa de palabras, utilizando el particionado 2, con SMOTE y sin él. . . . .	57
4.	Resultados de test con el modelo de bolsa de palabras, utilizando el particionado 2, con SMOTE. . . . .	57
5.	Validación del modelo de vectores de medias, partición 1. . . . .	59
6.	Testeo del modelo de vectores de medias, primera partición . . . . .	59
7.	Validación del modelo de vectores de medias, particionado 2 . . . . .	61
8.	Testeo del modelo de vectores de medias, particionado 2 . . . . .	61
9.	Resultados de diversas estrategias de aprendizaje, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas. . . . .	73
10.	Resultados de aplicación de técnicas de early stopping, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas. . . . .	73
11.	Resultados de aplicación de la técnica de Dropout, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas. . . . .	73
12.	Resultados de aplicación de la técnica de regularización de Ridge, sobre el conjunto de validación/test, arquitectura de 2 capas. . . . .	74
13.	Resultados de la configuración final, sobre el conjunto de test, arquitectura de 2 capas. . . . .	75
14.	Resultados de distintas configuraciones sobre optimizadores y tasa de aprendizaje, arquitectura de 3 capas, conjunto de entrenamiento y validación. . . . .	77
15.	Resultados de distintas configuraciones aplicando técnica de Early Stopping, arquitectura de 3 capas, conjunto de entrenamiento y validación. . . . .	77
16.	Resultados de distintas configuraciones aplicando regularización Dropout, arquitectura de 3 capas, conjunto de entrenamiento y validación. . . . .	77
17.	Resultados de distintas configuraciones aplicando técnica de Early Stopping, arquitectura de 3 capas, conjunto de entrenamiento y validación. . . . .	77
18.	Resultados de sobre el conjunto de test de la configuración final, arquitectura de 3 capas. . . . .	78

# 1. Introducción y objetivos

Internet se ha convertido en una poderosa plataforma, sobre la cual se sostienen gran parte de los servicios que utilizamos a diario. Influye en nuestras decisiones, nos ahorra tiempo a la hora de realizar ciertas tareas y nos mantiene conectados e informados.

La red nos ofrece noticias de diversa naturaleza, opiniones y versiones de un determinado hecho, narradas de forma distinta. Los usuarios han pasado a tener un carácter activo, asumiendo el rol de publicadores o difusores de información. Las redes sociales, por su parte, han concentrado la información en un solo lugar, ofreciendo una exploración de contenidos basada en los allegados e intereses del usuario.

Este escenario actual nos ahorra tiempo y nos facilita la experiencia de mantenernos informados, aunque supone un arma de doble filo. Aquellas noticias con un titular más impactante o un tema más controvertido tienen más probabilidades de ser compartidas en las redes, y una vez que el lector llega a ellas, queda a su criterio creer o no en la fiabilidad de ésta.

Las publicaciones más virales no son necesariamente las más fiables o veraces. Esto provoca que no se aproveche el potencial de los servicios web que utilizamos, podemos tener acceso a gran cantidad de información, pero sólo somos conscientes de una pequeña parte, que en muchas ocasiones se encuentra distorsionada o sesgada.

El acceso a noticias falsas o no fiables tiene un impacto en nuestras decisiones y nuestra postura ante diversos hechos. Se ha convertido en un problema tangible que está teniendo efectos negativos en la sociedad. Grandes plataformas como Facebook han sido acusadas de este problema, y han tenido que estudiar diversas formas de afrontarlo. Hemos perdido el control sobre los datos que compartimos y también de la calidad de los datos que llegan a nosotros. De esta forma, Internet se distancia de su objetivo inicial y se alcanza una desinformación en términos distintos [5].

La contrastación de noticias en la web es un problema complejo. Por un lado, la publicación de noticias se realiza en una magnitud tal que resulta imposible realizar esta tarea de una manera manual. Por otro lado, el proceso de verificar una noticia es otro problema en sí, en el que intervienen muchos factores. Es complicado definir qué es lo que consideramos como una noticia falsa y en qué criterios nos basamos para catalogar a una como tal.

Por tanto, es necesario disgregar el problema en distintas partes y abordarlo de forma incremental, desarrollando de esta forma herramientas que tengan un buen desempeño en una tarea concreta dentro del proceso de verificación de una noticia.

Una de las tareas interesantes que forman parte del proceso de verificación es el estudio de la posición del cuerpo de la noticia con respecto al titular. La posición viene definida principalmente a si el titular y cuerpo son contradictorios o consecuentes entre sí y en la relación de los hechos que se indican en el titular y los que luego se discuten.

El propósito del presente trabajo es diseñar e implementar un filtrador inteligente de noticias web que catalogue las noticias en base a su posición con respecto a

su titular. La plataforma consistirá en un lector RSS que ofrecerá a los usuarios las distintas noticias que existen en base a unas palabras clave, y las clasificará en función de su fiabilidad, en base al concepto de posición. La motivación de la herramienta es estudiar y ofrecer una forma de devolver en cierto grado la honestidad en la web.

La parte principal de esta herramienta es el motor inteligente que realizará una clasificación supervisada de las noticias. La solución más adecuada en estos casos es hacer uso de técnicas de Procesamiento de Lenguaje Natural y aprendizaje profundo. Las redes de neuronas artificiales han cobrado fuerza en este tipo de problemas, y en la actualidad disponemos de nuevas librerías y entornos de desarrollo, como TensorFlow (Google) [59], Spark (Apache) [63] o Scikit-Learn [54].

Este trabajo tiene dos objetivos principales diferenciados. En primer lugar, el aportar una herramienta para el filtrado de noticias, y, en segundo lugar, estudiar el potencial de las redes de neuronas profundas para tales fines. Los aspectos que se abordan y se exponen a lo largo del presente trabajo son los siguientes:

- Ventajas y desventajas del uso de Redes Neuronales Profundas para el Procesamiento de Lenguaje Natural.
- Búsqueda y preprocesamiento de los datos de entrada del modelo.
- Entrenamiento supervisado y creación del modelo utilizando Redes Neuronales Profundas.
- Análisis y validación del modelo que alimenta la plataforma.
- Diseño e implementación de la plataforma de filtrado de noticias basado en el modelo creado.

## 2. Estado del arte

En este capítulo se aporta la base científica relativa al proyecto realizado. Las dos primeras partes describen una panorámica sobre los modelos de representación utilizados y sobre las redes de neuronas artificiales. La tercera parte introduce *TensorFlow*, la herramienta utilizada para la construcción del modelo neuronal. Seguidamente, se introducen los mecanismos de optimización de redes de neuronas contemplados y se analiza la literatura existente sobre el tratamiento de datasets desbalanceados. La última sección aporta un estudio comparativo de los trabajos relacionados y expone la aportación al conocimiento que ofrece el presente trabajo.

### 2.1. Procesamiento de Lenguaje Natural. Representación de textos

El **Procesamiento de Lenguaje Natural (NLP)** es un área dentro de la Inteligencia Artificial que tiene como objetivo interpretar el lenguaje para llevar a cabo una tarea determinada. Actualmente se trata de una disciplina en auge, ya que es una pieza fundamental de tecnologías prometedoras como los sistemas cognitivos o los agentes conversacionales.

El objetivo de NLP es obtener toda la información posible a partir del lenguaje. La capacidad de interpretar consiste en que el sistema sea capaz de encontrar patrones o extraer información a partir de los datos. Entre las aplicaciones de uso podemos citar la traducción automática, la clasificación de textos o el análisis de sentimientos.

El **Procesamiento de Lenguaje Natural y Aprendizaje automático (Machine Learning)** son dos disciplinas muy relacionadas. El aprendizaje automático ofrece técnicas de aprendizaje computacional, mientras que el procesamiento de lenguaje Natural engloba a una serie de problemas que tienen como objetivo final interpretar el medio escrito. Se pueden diseñar sistemas que no sólo sean capaces de interpretar el lenguaje, sino que además sean capaces de aprender a partir de éste.

La riqueza expresiva que nos aporta el lenguaje hace que su procesado sea muy complejo. Para procesar el lenguaje el primer paso que tenemos que considerar es de qué manera vamos a tratar los textos. La representación de textos consiste en portar el lenguaje natural a una estructura de datos manejable por una máquina.

Decidir qué representación es la más adecuada no es un proceso trivial. Es necesario tener varias cuestiones en cuenta. En primer lugar, la representación debe ser capaz de mantener en su mayor medida la información que aporta el lenguaje natural. Con ello nos referimos a la semántica de los términos, y a la relación de cada palabra con su contexto, entre otros. Otro aspecto clave es si la representación es eficiente a la hora de almacenarla, y a la hora de utilizarla para su procesado y tratamiento.

La **semántica distribucional** es un área de investigación que estudia la representación de palabras para la extracción de propiedades cuantificables. Se fundamenta sobre la **hipótesis distribucional**, que sugiere que podemos inducir el



significado de un término a partir del contexto donde se utiliza. Una de las técnicas de representación de esta corriente es el modelo de **Bag Of Words** o bolsa de palabras.

### 2.1.1. Bag Of Words (BOW)

La representación de textos más extendida es **Bag-of-Words** [21]. Es sencilla y ofrece unos resultados razonables en el procesamiento del lenguaje natural. El modelo de Bag of Words toma como entrada un conjunto de textos y genera una representación vectorial de longitud constante. La representación resultante refleja las ocurrencias de cada término a lo largo de los documentos que forman parte del corpus.

El primer paso para generar un modelo de bolsa de palabras es tokenizar el documento por palabras. Seguidamente se aplica un método de conteo de las palabras y se normaliza el conteo resultante. Usualmente se utiliza el producto de dos estadísticas, TF - IDF (Term Frequency - Inverse Document Frequency), que en conjunto reflejan la relevancia de la palabra en el documento, es decir, la medida de información que aporta. **TF-IDF** considera como relevantes y asigna un peso mayor a aquellos términos que tienen una ocurrencia alta, pero aparecen en pocos documentos. Esta fórmula nos permite discriminar las conocidas como **stop words**, palabras que aparecen muy frecuentemente (artículos, preposiciones, etc) a lo largo de los documentos, pero que no aportan significado en sí a ninguno en particular.

$$Tf - Idf(t, d, D) = tf(t, d) * idf(t, D)$$

*(t: término, d: documento, D: conjunto de documentos del corpus)*

Para el conteo de términos (tf) podemos optar por diversos métodos de peso, como el conteo en crudo, conteo binario o a la frecuencia del término (en base a la longitud del documento). Para el cálculo de la inversa de la frecuencia (idf) disponemos también de varias opciones, desde la versión básica, a modificaciones que intentan suavizar el valor de la métrica o versiones probabilistas. La configuración básica de la fórmula sería la siguiente:

$$Tf - idf(t, d, D) = f(t, d) * \log N / nt$$

Esta representación es sencilla de construir, pero no refleja toda la riqueza del texto. Una de las principales desventajas es que no se tiene en cuenta el orden de las palabras a la hora de generar la representación. Esto hace que se pierda el aspecto semántico de los términos. Otro de los inconvenientes es la gran dimensionalidad del modelo resultante. Este fenómeno recibe el nombre de **dispersidad de datos** o **data sparsity**. Esto hace más complejo el cómputo del modelo y su manejo a la hora de ser utilizado en un sistema de Machine Learning.

Bag of Words no deja de ser una representación de la distribución de palabras en el texto desde un punto de vista estadístico, sin tener en cuenta el significado tras cada palabra o la relación de una palabra con su contexto. Existe una variación que

intenta paliar este aspecto, y se trata del modelo **Bag-N-Gram**. En esta variante realizamos el conteo de las palabras por grupos. Por ejemplo, en el modelo Bi-Gram, consideramos las palabras por tuplas y generamos una representación en base a la ocurrencia de estas tuplas.

Una opción para lidiar con la dispersidad de los datos es aplicar una **reducción de dimensionalidad** sobre la representación, por medio de técnicas como la **descomposición en valores singulares** (SVD, Singular Value Decomposition). Este método es una forma de factorización de matrices, que genera una matriz reducida que mantiene la varianza de la original. Es un método similar al método estadístico PCA (Principal Component Analysis).

Este modelo sigue siendo válido para gran parte de aplicaciones de procesamiento de lenguaje natural, como por ejemplo clasificación de textos. Pero existen ciertas aplicaciones como la traducción automática o la recuperación de información que se verían beneficiadas de representaciones que tengan en cuenta más factores del lenguaje. Esta motivación ha dado lugar a la investigación en nuevas representaciones como son los **modelos basados en redes de neuronas**.

### 2.1.2. Modelos basados en redes de neuronas

Con el renacimiento de las redes neuronales artificiales han surgido nuevos enfoques para construir modelos de representación de textos. Estos modelos pertenecen a la categoría de **métodos predictivos**, que, a diferencia de los métodos de conteo de la semántica distribucional, utilizan la predicción de términos en el documento en lugar de su conteo como base para crear una representación.

El entrenamiento del modelo se realiza de forma iterativa, hasta que se alcanza un cierto número de iteraciones o nos encontramos por debajo de un umbral de error aceptable. El objetivo del entrenamiento es maximizar una función objetivo, como la probabilidad logarítmica de ocurrencia de un término dado. El proceso genera una representación de **vectores continuos en el espacio** (Vector Space Models), que sitúan los términos en base a su significado, frente a los métodos de conteo que asignan una representación de vectores discreta a partir de las ocurrencias. Los vectores continuos nos permiten algo tan valioso como establecer relaciones entre los términos y obtener una visión global y unificada del vocabulario en sí.

Dos modelos de representación dentro de esta corriente son los **Paragraphs Vectors** y los **Embedding Vectors**. Estos modelos tienen en común el hecho de partir de una red de neuronas con una arquitectura simplificada con respecto a otras soluciones (no se emplean redes neuronales profundas). La ventaja de esta decisión es que los modelos serán menos complejos en términos de cómputo. Pese a partir de una arquitectura sencilla, se consiguen unos resultados al mismo nivel o mejorando el estado del arte alcanzado por otras soluciones.

### 2.1.3. Embedding vectors. Word2Vec

Los **Embedding Vectors** [35] surgen con la premisa de realizar un análisis más específico de los documentos y obtener una representación orientada a las palabras.

Se trata de un modelo similar a Paragraph vectors, pero en este caso eliminando la representación de los párrafos en el documento como feature para la predicción del siguiente término. El nombre de esta representación se debe a que cada palabra tiene “embebida” un vector que la representa, y expresa su relación con el contexto.

**Word2Vec** es la implementación de este modelo. Se encuentra disponible en distintas plataformas como gensim [49], TensorFlow [59], Deeplearning4J (Deep Learning for Java) [55] o Spark [57]. Se trata de una red de neuronas de dos capas que procesa texto. A partir de un corpus obtiene un conjunto de vectores que representa los términos. Bajo este enfoque, las palabras se consideran estados discretos, y la conexión entre dichos estados representan una relación de contexto. La representación se construye a partir de las conexiones que presentan una mayor probabilidad.

Lo más novedoso de este tipo de representaciones es que se ha demostrado que podemos hacer preguntas al modelo sobre la **relación entre términos** (razonamiento análogo). Esto es posible aplicando operaciones de suma/resta sobre los vectores de representación, traducándose en la siguiente expresión:

$$“Portugal es a Lisboa lo que España a X” \quad \text{vec}(\text{“Portugal”}) + \text{vec}(\text{“Lisboa”}) = \text{vec}(\text{“España”}) + \text{vec}(X)$$

A partir de la expresión se calcula el vector de representación y se obtiene el término que se encuentra situado en ese punto del espacio continuo. Si ningún término satisface la expresión se escoge aquel término cuyo *embedded vector* se encuentre más cercano según la distancia de cosenos.

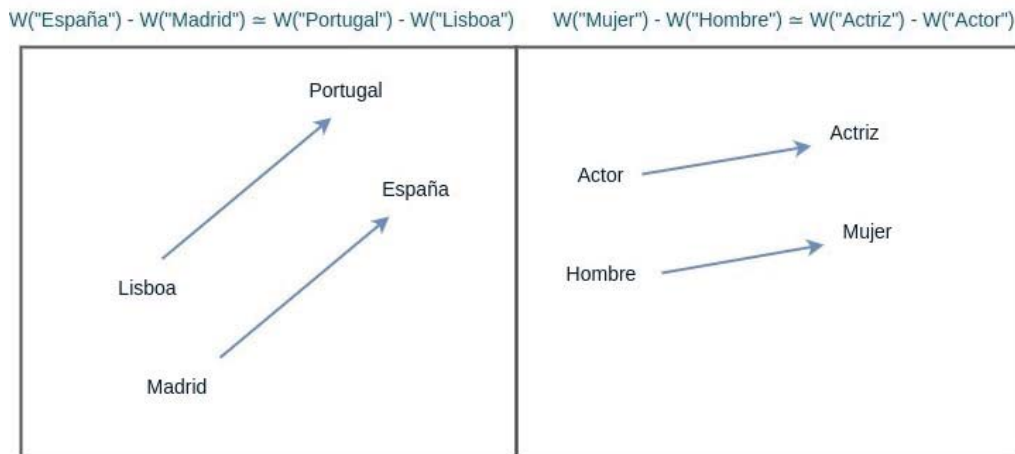


Figura 1: Ejemplos de analogías que pueden ser obtenidas a partir de un modelo de word embeddings

El modelo de embedding vector presenta dos arquitecturas distintas: **Continuous Bag of Words (CBOW)** y **Skip-gram**. Se diferencian entre sí por las features de entrada y el objetivo a predecir (target). En el modelo de Continuous Bag of Words (CBOW), utilizamos el contexto para inferir el término al que corresponde, y en el método Skip-gram tomamos como origen un término para obtener

el contexto. En ambos casos se realiza una regresión logística (clasificación discreta) para distinguir el término objetivo de un posible término erróneo y se utiliza retropropagación para reentrenar el modelo.

**Continuos Bag Of Words** En el modelo de **Continuous Bag of Words** inferimos la palabra actual a partir del contexto. Para ello tomamos las palabras anteriores y posteriores a la palabra a predecir. Consiste en un modelo similar a la versión de memoria distribuida de *paragraph vector*, pero esta vez sin tener en cuenta la representación del párrafo para la predicción de un término del párrafo.

La red neuronal que utilizamos tiene tres capas: una de entrada, una de proyección y otra de salida. La capa de entrada recoge las palabras del contexto, y éstas pasan a ser proyectadas en la misma posición. Se realiza una media sobre el vector de representación de cada palabra para predecir el término resultante.

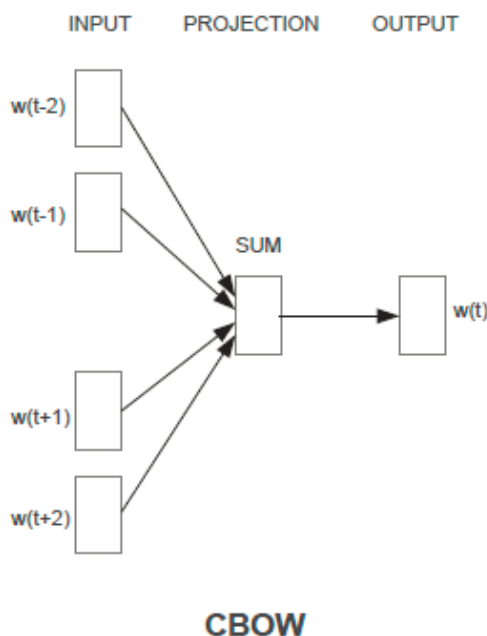


Figura 2: Arquitectura de CBOW [35]

La red neuronal que utiliza CBOW es un **clasificador softmax**. La función de *softmax* es una versión de la función logística que redimensiona un vector de entrada a otro de dimensión fija y valores normalizados. Para optimizar la implementación de este modelo se utiliza habitualmente la función de **hierarchical softmax (softmax jerárquico)**. La versión de hierarchical hace uso de un árbol binario para la clasificación, que reduce el número de salidas a evaluar ( $\log_2(w)$ ), por lo que es menos costoso computacionalmente [35].

Se ha demostrado que CBOW tiene un desempeño mejor con respecto a skipgram sobre datasets más pequeños, ya que con el segundo es necesario disponer de una gran dataset para hacer un buen modelo.

**Skip-Gram** La aproximación de **Skip-gram** supone el enfoque inverso con respecto CBOW, ya que en este caso partimos de un término para predecir los términos que se encuentran en el contexto. Para ello, alimenta la red de neuronas con tuplas del tipo (*término de entrada, término del contexto*) en base al tamaño de entrada del modelo. Mayores tamaños de ventana generan un mayor número de tuplas de entrada. A partir de las tuplas de entrada la red calcula las probabilidades que se le da a cada tupla. Cuantas más ocurrencias de una misma tupla de entrada se den, mayor probabilidad se asignará a la tupla.

Una vez entrenado, el sistema tomará como entrada la representación vectorial de un término, que tendrá la forma de un vector de dimensión  $N$  (número de términos totales del dataset). La salida que generará es un vector de las mismas dimensiones con las probabilidades de ocurrencia del término con cada uno de los términos del corpus. Los pesos de las neuronas de la capa oculta caracterizan la distribución del término y su contexto.

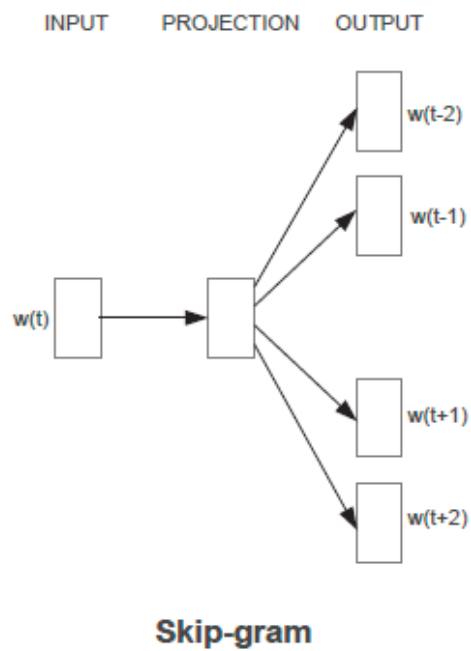


Figura 3: Arquitectura de Skip-gram [35]

En **Skipgram** se utiliza una variante del modelo de softmax, llamada **Noise Contrastive Estimation (NCE)**[31]. Al partir de un único término para realizar la predicción corremos el riesgo de que los términos muy frecuentes (stopwords) introduzcan ruido o dificulten el entrenamiento. El propósito de NCE es discriminar estos términos.

Skipgram ofrece unos resultados ligeramente más precisos con respecto a CBOW, si utilizamos un dataset de entrada lo suficientemente grande.

**Paragraphs Vectors. Doc2Vec** El modelo de Paragraphs Vectors [31] es un framework de entrenamiento no supervisado que genera una representación de longitud fija a partir de cualquier párrafo de un documento de entrada.

En este enfoque, concretamente en la versión de **memoria distribuida (Distributed Memory Model of Paragraph Vectors)**, se construyen dos representaciones: una para el documento (D) y otra para las palabras del documento (W). Cada columna de la matriz D es la representación de un párrafo, y cada matriz de W es la representación de un término. El entrenamiento del modelo consiste en alcanzar una representación que sea capaz de predecir la siguiente palabra de una secuencia de términos a partir de sus representaciones. Los dos vectores de representación se ajustan de forma iterativa de acuerdo con el algoritmo de retropropagación del gradiente, utilizado clásicamente en las redes neuronales artificiales.

La idea tras este modelo es que las palabras que aparecen en un documento dado dependen en gran parte de la temática de éste. Para predecir el siguiente término de un párrafo, tomamos como entrada la propia representación del párrafo y de los vectores correspondientes a las palabras del contexto (términos en los alrededores, según un tamaño de ventana fijado) para predecir el próximo término. El objetivo de mantener una representación para los párrafos es tener una visión de la temática y complementa al contexto inmediato de términos que se tienen en cuenta.

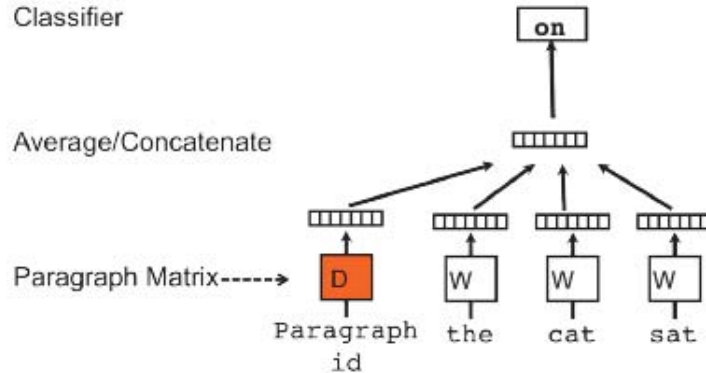


Figura 4: Diagrama sobre la predicción de términos en base al Paragraph Vector y los términos del contexto [31]

Las mayores ventajas de los paragraphs vectors es que no se requiere de datos etiquetados para el entrenamiento y que tienen en cuenta la semántica y orden de ocurrencia de las palabras. Esto constituye una mejora con respecto al modelo de Bag-n-gram, que pese a tener en cuenta cierto contexto, da lugar a representaciones con mayor dimensionalidad [31].

Existe una variación del algoritmo citado en la que partimos únicamente de la representación del párrafo para predecir las palabras del contexto. Esta variante recibe el nombre de Distributed Bag Of Words(PV-DBOW)[31].

La implementación de estas ideas da lugar al modelo **Doc2Vec**, que se encuentra disponible en frameworks de NLP como gensim [50].El modelo de paragraph vectors

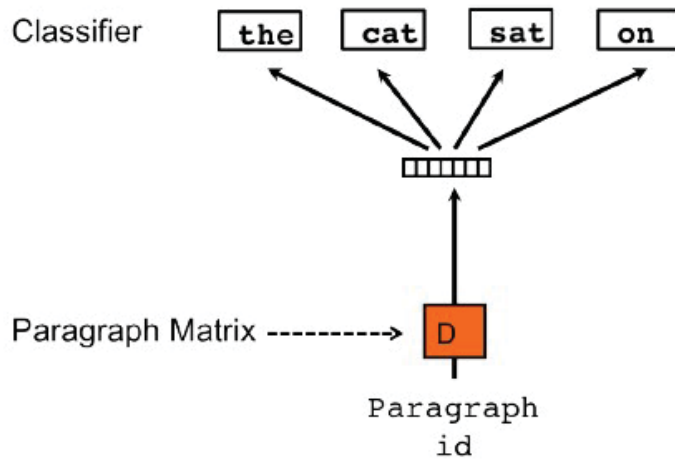


Figura 5: Variante de Bag Of Words distribuida [31]

resulta de utilidad cuando queramos tratar los documentos en su conjunto, por su temática o sentimiento. Un ejemplo de aplicación propuesto es el análisis de sentimientos sobre un conjunto de críticas de películas [31]. Existe otro tipo de modelos similar a los **Paragraph Vectors**, llamado **Embedding vectors**, pero esta vez enfocados a obtener una representación términos del documento.

## 2.2. Redes de neuronas artificiales

Las redes de neuronas artificiales surgieron a partir de los descubrimientos en el pasado siglo sobre las redes neuronales biológicas y la teoría conexionista. Dicha teoría establece que las neuronas biológicas se disponen en capas y que el aprendizaje humano surge gracias a las sinapsis o conexiones entre las neuronas. Una neurona genera una señal en base a los estímulos que recibe y las conexiones entre cada neurona se hacen más fuertes si reaccionan de la misma forma, es decir, generan la misma salida, ante un determinado estímulo.

Las redes neuronales se encuentran especializadas en llevar una tarea concreta. La disposición de las neuronas biológicas en extensas redes les permite hacer cálculos y tareas complejas como llevar a cabo funciones motoras o cognitivas, por lo que a raíz de esta premisa surgió el interés de plasmar este comportamiento como un modelo computacional.

La primera propuesta de **redes neuronales artificiales** (RNA, o ANN en inglés) surgió en 1943, de la mano del neurofisiologista **Warren McCulloch** y el matemático **Walter Pitts**. Si bien las redes de neuronas artificiales partieron como un modelo fiel a su análogo biológico, durante su evolución se han ido distanciando de éste, para adaptarse más a los casos de aplicación o restricciones computacionales.

En las redes de neuronas artificiales representamos la relevancia de una conexión asignándole un peso. El entrenamiento es una clasificación supervisada que consiste en el ajuste iterativo de estos pesos, inferidos a partir de un conjunto de datos de



entrenamiento, que representan los datos de entrada y la salida esperada.

Para el entrenamiento se aplica el **algoritmo de retropropagación**, que en cada iteración parte de una predicción del dato de salida (forward pass) a partir de un dato de entrada de entrenamiento. En base a la predicción calcula desde la capa de salida a la capa de entrada los errores sucesivos con respecto a la salida esperada (reverse pass). Finalmente computa el gradiente descendiente con respecto del error, es decir, reconfigura los pesos de tal forma que se minimice el error.

El perceptrón simple, o **Linear Threshold Unit (LTU, 1977)** es la representación de una neurona, se encuentra conectado a las entradas del sistema y genera una salida en consecuencia. Para obtener la salida primero realiza una suma ponderada de las entradas (en base al peso de la conexión) y en base a esta aplica una función de paso o de activación que define la salida final del sistema. Existen varias funciones de activación, entre las que se encuentra la *función hipérbolica tangente* ( $\tanh$ ) o la función *ReLU* (*Rectified Linear Unit, función rectificadora*).

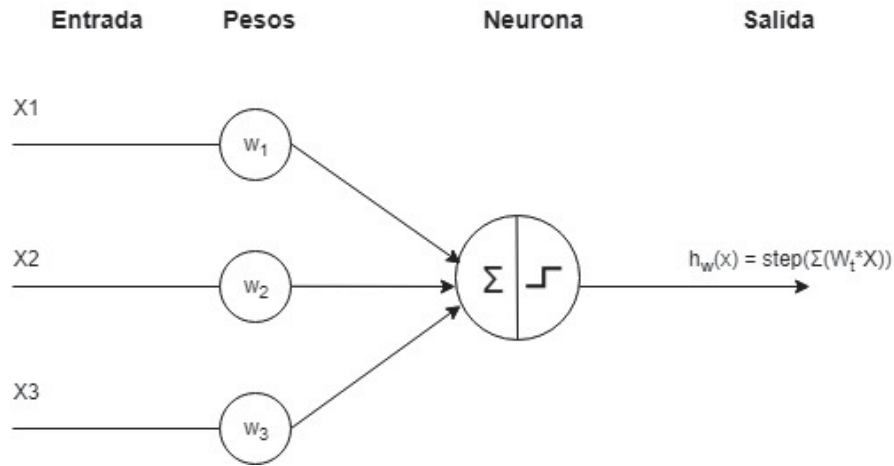


Figura 6: Perceptrón Simple (Linear Threshold Unit)

El perceptrón simple se puede utilizar para construir estructuras más complejas, como una red de neuronas de una capa. Esta arquitectura es capaz de modelar problemas de clasificación, siempre y cuando los datos de entrada sean linealmente separables. Esta arquitectura no es capaz de detectar la probabilidad de pertenencia a las clases de salida del modelo, por lo que no resulta una mejora con respecto a otros modelos clásicos como el regresor logístico, que sí es capaz de hacerlo.

Para construir modelos más avanzados es necesario utilizar más capas. La arquitectura de neuronas más extendida es el perceptrón multicapa, formado por una capa de entrada, una o varias capas ocultas y una capa de salida. La capa de entrada toma los parámetros o features del entrenamiento y los hace pasar a la capa siguiente. Las capas intermedias extraen características o patrones a partir de las entradas y terminan propagando unas salidas a la última capa.

El número de neuronas de la capa de salida se encuentra condicionado por el tipo de modelo. En un problema de regresión lineal tendremos una neurona de salida, en una clasificación binaria una neurona que aplica una función de activación, y



en una clasificación una neurona por clase, sobre la que se aplica una función de softmax, que determina la probabilidad de pertenencia a cada clase. Si las clases son excluyentes se puede utilizar una función de softmax compartida para todas las neuronas de salida.

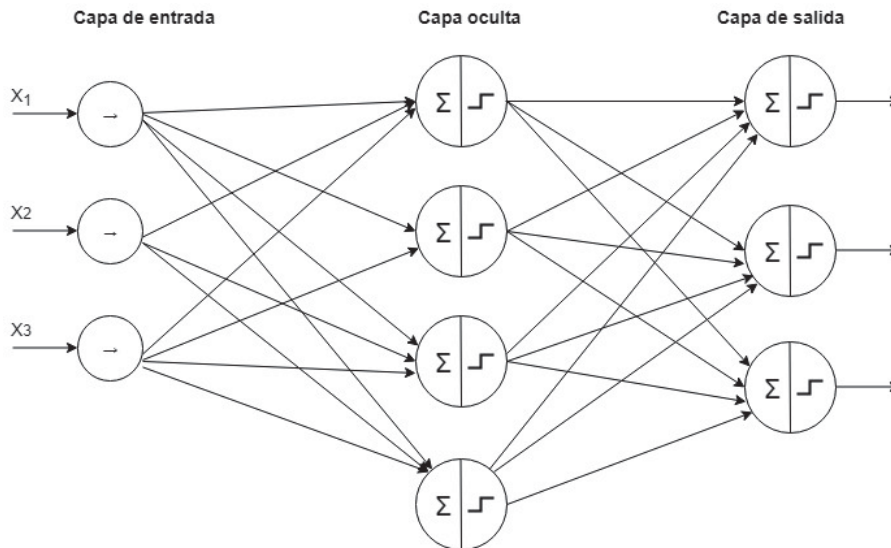


Figura 7: Perceptrón Multicapa (Multilayer Perceptron, MLP) con una capa oculta

Todas las capas de neuronas salvo la capa de entrada se encuentran conectadas a todas las neuronas de la capa anterior. La salida que genera cada neurona se calcula en dos fases. En la primera se realiza una media ponderada de las entradas, en función de los pesos de cada conexión. En la segunda fase se toma el valor de la media y se aplica una **función de paso** (function step) para calcular la salida de la neurona.

Un perceptrón multicapa es lo suficientemente potente para modelar problemas de regresión o clasificación, pero en otro tipo de problemas necesitamos arquitecturas de red más complejas. Las redes de neuronas profundas son la base de **Deep Learning**, y son aquellas que tienen más de una capa oculta. Dichas capas se disponen una a continuación de la otra.

El objetivo de la red es encontrar una función que modele los datos de entrada pertenecientes a un problema o dominio concreto. En problemas de regresión la función determina el valor predicho para la salida, en base a las entradas definidas. En el caso de la clasificación cada una de las neuronas de salida determinan una función de activación que divide el plano de datos de entrada entre los datos que pertenecen a dicha clase de los que no. En ocasiones resulta de utilidad desplazar dicha función, para mejorar la precisión de la clasificación.

Para ello se añade un factor ajeno a los datos de entrada, llamada **neurona de bias**. Esta neurona se puede situar en la capa de entrada o en cualquiera de las capas ocultas y se encarga de propagar un valor "1.<sup>a</sup> la siguiente capa. Tras el entrenamiento de la red se establecerá los pesos de las conexiones de dicha neurona, por lo que podrá propagar valores variables a cada neurona de la capa posterior.

El perceptrón multicapa sigue una arquitectura de **red neuronal hacia delante** (Feed Forward Network, FNN) y es capaz de modelar cualquier problema que siga una función continua. Esta afirmación se conoce como el **teorema de aproximación universal** [25], que constata que la arquitectura de la red en sí, por encima de la selección de la función de activación, es la que en gran medida le da tal potencial a las redes neuronales.

Una arquitectura tan sencilla como una capa de entrada, una capa oculta y una capa de salida nos brinda la posibilidad de representar de manera aproximada un gran abanico de problemas. No obstante, para casos más complejos se requiere de una red más elaborada.

Una red formada por más capas ocultas puede extraer más información de los datos y establecer patrones de comportamiento que se propagan a las capas posteriores. Por ejemplo, en una red para el reconocimiento de imágenes, las capas iniciales reconocerían características de bajo nivel (segmentos), las intermedias formas simples (polígonos) y las posteriores reconocerían la composición de estas formas (objeto).

Una **red neuronal profunda** es aquella que está formada por más de una capa oculta. Una red profunda converge mejor que una simple, es decir, alcanza una solución estable en menor tiempo. Además, estas arquitecturas se pueden reutilizar como base para modelar casos más complejos. De esta forma, se puede crear un modelo posterior que tome como datos de entrada la salida de una red que se dedique a modelar un caso concreto (por ejemplo, detección de formas).

La mayor complejidad de las redes neuronales reside en descubrir la arquitectura y aquella configuración que sea capaz de aprender de los datos de entrada y concluir con una función que los aproxime. Los hiperparámetros que se deben fijar son el número de capas ocultas, el número de neuronas por capa y la función de activación de cada una, por lo que las redes neuronales son muy flexibles, pero resulta complejo encontrar la configuración más adecuada.

En cuanto a la arquitectura de red neuronal, se encuentran las siguientes:

- **Redes Neuronales hacia Delante (FeedForward Neural Networks, FNN).** En esta arquitectura el flujo de información va siempre hacia delante. Existe una primera capa oculta conectada a una capa de entrada, que hace pasar los datos de entrenamiento. Las salidas de una capa son las entradas de la capa posterior, hasta llegar a la capa de salida.
- **Redes Neuronales Convolucionales (Convolutional Neural Networks, CNN).** Las redes convolucionales están bioinspiradas en las redes que conforman la corteza visual primaria (V1), que se encarga del reconocimiento de patrones y procesamiento de información sobre los objetos reconocidos. Las redes artificiales convolucionales se utilizan principalmente en tareas de visión artificial, para la clasificación y reconocimiento de imágenes, aunque se pueden utilizar en otros ámbitos como la clasificación de textos.
- **Redes Neuronales Recursivas (Recursive Neural Networks, RNN).** En esta arquitectura se aplican recursivamente el mismo conjunto de pesos

sobre una estructura para realizar una predicción estructurada sobre una variable.

- **Redes Neuronales Recurrentes (Recurrent Neural Networks, RNN).** Este tipo de arquitectura está enfocada al tratamiento de flujos de datos, es decir, cubre problemas en los que el orden o secuencialidad de los datos es relevante a la hora de modelar un problema. A diferencia de las redes neuronales hacia delante, las neuronas poseen conexiones de realimentación.
- **Autocodificadores (Autoencoders).** Los autoencoders se utilizan para extraer de manera no supervisada características de los datos. Dada una entrada tratan de generar una representación eficiente de la entrada. Esta arquitectura se compone de dos partes, el codificador, que construye una representación interna a partir de los datos de entrada, y el decodificador, que a partir de esta genera una salida. Los autocodificadores se usan para tareas de reducción de dimensionalidad, generación de datos y como generadores de características que se utilizan como entrada para otros modelos de aprendizaje.
- **Memoria a corto plazo (LSTM).** Se trata de una arquitectura basada en redes neuronales recurrentes enfocada al tratamiento de secuencias de datos temporales. Las células LSTM se encuentran formadas por un puente de entrada (input gate), uno de salida (output gate) y un puerto que se encarga de descartar datos (forget gate).

## 2.3. TensorFlow

TensorFlow es un framework de computación numérica enfocado a la creación de modelos de Redes de Neuronas Artificiales. Es una librería de código abierto desarrollada por Google, y publicada en noviembre de 2015. Actualmente se encuentra disponible en Python, C++, Java, Go, Haskell y R, y se utiliza en plataformas de la empresa como Search, Gmail, Translate y Youtube.

El precedente de TensorFlow es DistBelief, publicada en 2012, y presentada en el artículo “Large scale Distributed DeepNetworks” [11]. Distbelief era una herramienta propietaria, concebida para ejecutarse en los datacenters de Google, por lo que su mayor ventaja era su escalabilidad y su compatibilidad con herramientas de la empresa como Inception (modelo de reconocimiento de imágenes) o RankBrain (motor de búsquedas).

No obstante, disponía de sus limitaciones, ya que no tenía un buen rendimiento en su ejecución con GPU, una característica que resulta muy provechosa para el entrenamiento de modelos de Machine Learning, en el que se tienen que realizar cálculos sobre cantidades ingentes de datos. Otra de sus principales desventajas era su falta de flexibilidad en cuanto a la creación de modelos de Machine Learning más elaborados, como los modelos de secuencia (RNN), o las arquitecturas de aprendizaje reforzado.

TensorFlow surgió como un rediseño de DistBelief, partiendo de sus fortalezas y añadiendo más expresividad en la creación de modelos, así como un soporte más

amplio. Es una plataforma abierta, en desarrollo constante y que crece en base a las necesidades de la comunidad. Trata de dar cabida desde aquellos usuarios que simplemente necesitan un modelo básico de Machine Learning hasta aquellos con problemas más exigentes donde el rendimiento es clave, o necesitan un modelo muy especializado a su caso de uso.

TensorFlow expone varias capas de abstracción, que se adaptan al nivel de cada desarrollador que utilice la plataforma. Las capas de alto nivel están orientadas a usuarios que se inician en Machine Learning o que no requieren crear un modelo muy específico. Las capas más bajas nos abren un gran abanico de posibilidades de configuración, ofreciendo primitivas para la creación de modelos completamente personalizados.

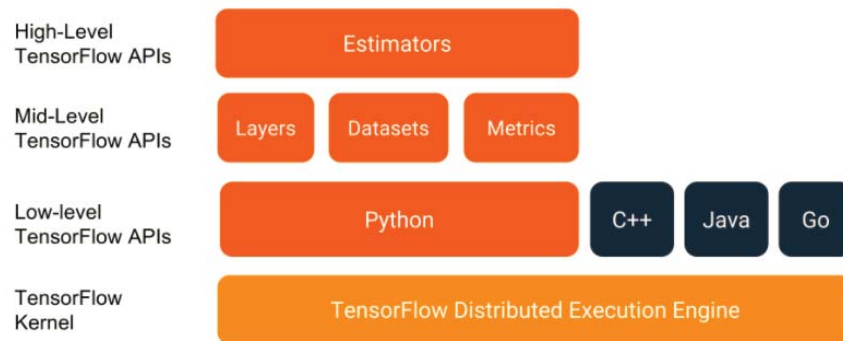


Figura 8: Pila de APIs ofrecidas por TensorFlow [60]

Los usuarios menos exigentes pueden hacer uso de las APIs de alto nivel como **Estimator**, que brindan la posibilidad de crear modelos y ejecutarlos de manera distribuida, todo ello utilizando primitivas de alto nivel.

Los estimadores son una serie de recursos que encapsulan tareas de entrenamiento, evaluación, predicción y exportado de modelos. En base a estos se pueden construir modelos preconstruidos con un esfuerzo menor. Esta semántica de acciones oculta al usuario de tener que construir el grafo de computación directamente, o en otros términos, le permite crear un modelo desde un punto de vista conceptual.

En los apis de nivel medio se ofrecen métodos intermedios entre la definición a nivel conceptual y la manipulación directa de grafos. El API de **Layers** [61] ofrece operadores para crear las capas de la red neuronal. Ofrece un gran número de arquitecturas de conexión, como capas completamente conectadas o capas convolucionales, y capas de computación habituales como capas que aplican la función ReLu o la función de SoftMax.

Por su parte el api de **Datasets** [61] proporciona métodos comunes para el manejo de las fuentes de datos que se utilizan en los modelos. Entre las operaciones disponibles se encuentra la carga de modelos, la división en grupos (de cara al entrenamiento o a la validación cruzada de modelos), y algunas operaciones de manipulación como el agitado o repetición de muestras.

El módulo de **Metrics** [61] está formado por funciones de cálculo de métricas como accuracy, precision o recall. El cálculo de métricas tiene tres fases: inicialización, agregación (actualización del valor actual de la métrica) y finalización (cómputo del valor final de la métrica).

Para aquellos usuarios que deseen tener un control directo de los modelos, sin abstracciones, existe un api de bajo nivel para la definición de los modelos. Además, en el kernel se ofrecen herramientas muy interesantes para optimizar la ejecución como **XLA (Accelerated Linear Algebra)** [13], un compilador con instrucciones de ensamblador especiales para realizar operaciones sobre grafos. XLA soporta dos modos de compilación: *JIT (Just in time)* y *AOT(Ahead of time)*. Esta herramienta resulta muy provechosa en modelos que requieren computaciones de grandes volúmenes de datos, o se ejecutan en entornos de ejecución más modestos, como son las plataformas móviles. XLA se encuentra actualmente en versión alpha.

El **Core Distribution Engine** es la capa entre la capa hardware y la API de desarrollo. Está desarrollado en C++, gestionar la ejecución distribuida de los modelos y de lidiar con la gestión de las diversas plataformas de Hardware de ejecución. Tensorflow es capaz de ejecutar sobre CPUs, GPUs, TPUs (Tensor Processing Unit), y plataformas móviles como smartphones y Raspberry Pi. Además, ofrece soporte para TensorFlow está pensada para ejecutarse en plataformas en la nube como Google Cloud Platform, y permite entornos de ejecución distribuidos en varias máquinas.

Finalmente, aunque la plataforma se concibió para implementar redes de neuronas artificiales, también tiene puesto el foco en proporcionar implementaciones de modelos de machine learning clásicos, y la lista de algoritmos soportados va creciendo con el tiempo. Actualmente da soporte para la creación de modelos Kmeans para clustering, Support Vector Machines para clasificación, y random forests para labores de clasificación y regresión, entre otros.

### 2.3.1. Definición de grafos de computación

En Tensorflow la computación está enfocada a grafos. Un grafo computacional describe la secuencia de operaciones que se realizan sobre los datos de entrada, para generar una salida. En un grafo se representa el orden y la relación que existe entre las operaciones de una determinada función.

Los dos objetos fundamentales de un grafo son las operaciones y los tensores. Las operaciones son los nodos del grafo y definen una transformación concreta sobre el dato de entrada. Los tensores son la representación de los datos de entrada (extremos del grafo), y pueden ser un array o un escalar. Los tensores se representan internamente por medio de arrays numpy de 0 a N dimensiones.

Un grafo de cómputo puede estar formado por los siguientes tipos de nodos [62]:

- **Nodos de operación:** Los nodos de operación toman como entrada 0 a N tensores y generan como salida de 0 a N tensores. El api proporciona un gran número de operadores, que van desde los operadores aritméticos simples a operadores propios de machine learning, como el optimizador del gradiente descendiente [61].

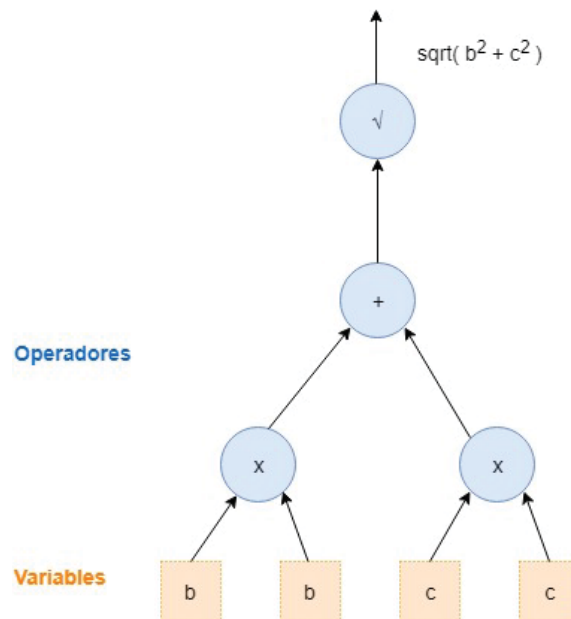


Figura 9: Ejemplo de grafo de computación, que aplica el Teorema de Pitágoras

- **Nodos de variables (`tf.Variable`) y constantes (`tf.Constant`):** son la representación en el grafo de los tensores. El API de TensorFlow proporciona varias operaciones para generar constantes como `random` (genera un tensor de números aleatorios), `constant` (valor fijado) u `ones` (genera un tensor de unos).
- **Nodos de inicialización de variables (`tf.global_variables_initializer`):** Es el operador que inicializa todas las variables globales declaradas en el grafo. Debe ejecutarse para que todas las variables tomen el valor de partida.
- **Nodos de guarda (`tf.saver`):** se trata de un operador especial que permite guardar y recuperar variables o modelos de cómputo. Dichos elementos se guardan en un fichero con el nombre especificado. Resultan muy útiles para crear puntos de salvaguarda (checkpoints) durante la fase de entrenamiento, y para guardar en memoria el modelo de Machine Learning final.
- **Nodos Placeholders (`tf.placeholder`):** reservan un espacio de memoria para un valor de entrada para el modelo. Habitualmente se utilizan para declarar las neuronas de la capa de entrada, que recibirán las muestras de entrenamiento, en forma de tensores.

Los nodos que se declaran pertenecen al grafo por defecto, aunque podemos asignarlos explícitamente a otro. Todos los valores asignados a un determinado nodo son reseteados entre ejecuciones del grafo, con excepción de las variables, que pueden ser mantenidas por una sesión a lo largo de varias ejecuciones.

Un programa de tensorflow tiene dos fases. La primera es la fase de construcción, y en ella se define el grafo de computación. Para ello se declaran las variables de

entrada, el número de capas de la red, las conexiones y se declara la función de coste y la función de optimización de la red.

La siguiente fase es la fase de ejecución y en ella se ejecuta el grafo. Los nodos del grafo no se computan hasta que se lanza la ejecución explícitamente dentro de una sesión de ejecución. La sesión representa la conexión con el programa del usuario y el entorno distribuido de ejecución. En la fase de ejecución se entrena el modelo creado. Para ello se declara dentro de la sesión un bucle que realiza el entrenamiento iterativamente.

Se pueden agrupar nodos que formen parte dentro de un determinado contexto, como las neuronas de una capa de la red neuronal, dentro de lo que se conoce como **name scopes**. El uso de name scopes [61] es una buena práctica que mejora la legibilidad del grafo y facilita el manejo y depuración de este. También es una práctica común para ejecutar cada sección del grafo en un dispositivo distinto, definido por el desarrollador.

## 2.4. Técnicas de optimización y regularización de Redes neuronales.

En la presente sección se realiza una descripción de la literatura existente sobre métodos de optimización y regularización empleados en este trabajo. La descripción se acompaña de un breve comentario sobre su uso en este trabajo.

### 2.4.1. Métodos de optimización.

El clasificador que se presenta en este trabajo utiliza una arquitectura de red hacia delante. Para su desarrollo y optimización se han tenido en cuenta distintas estrategias de optimización, configuración de capas y neuronas, y métodos de regularización.

Además del Gradiente Descendiente Clásico, se ha tenido en cuenta la optimización del **Gradiente Descendiente con Momentum**. El método momentum tiene una eficacia validada sobre arquitectura de redes profundas, en arquitecturas como la red neuronal hacia delante o la red neuronal recurrente [58].

El mecanismo de optimización momentum tiene como propósito acelerar el alcance de la convergencia al mínimo local, reduciendo el error en la función de coste. Postulado por Boris Polyak en 1964, introduce el concepto de velocidad en la actualización del gradiente descendiente [44]. Para ello utiliza un vector de velocidad que acumula las reducciones realizadas a lo largo de la optimización de la función objetivo (gradiente descendiente de las iteraciones anteriores). Los parámetros del optimizador son la tasa de aprendizaje y el momentum, un factor que simula la fricción a lo largo de la optimización, para evitar que el vector de aceleración se incremente demasiado. Estudios recientes proponen un valor de momentum en el rango de 0.9 a 0.99 [58].

Existe un método similar conocido como **Gradiente Acelerado de Nesterov** [38], y se ha validado como un optimizador con una convergencia más rápida con



$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t)$$

$$\theta_{t+1} = \theta_t + \mu v_t - \varepsilon \nabla f(\theta_t)$$

Figura 10: Expresiones para el Momentum Clásico [58]

respecto al método clásico. El método de Nesterov calcula el vector de velocidad actualizando el vector con respecto a una posición posterior de la función de coste, con el propósito de prevenir oscilaciones en la convergencia y acelerar el alcance del mínimo local.

$$v_{t+1} = \mu_t v_t - \varepsilon_t \nabla f(\theta_t + \mu_t v_t)$$

$$\theta_{t+1} = \theta_t + \mu_t v_t - \varepsilon_t \nabla f(\theta_t + \mu_t v_t)$$

Figura 11: Expresiones para el Gradiente Acelerado de Nesterov [58]

En este trabajo se emplea la versión clásica del algoritmo, con el valor de momentum recomendado (0.9), para el uso de dos capas ocultas. Para arquitecturas de tres capas se ha experimentado con la variante de Nesterov con el propósito de disminuir el tiempo de entrenamiento del modelo.

#### 2.4.2. Técnicas de regularización en Redes de Neuronas

Las técnicas de regularización tienen como objetivo corregir escenarios de overfitting. Para ello, aplican penalizaciones en la fase de entrenamiento de los modelos, para evitar un sobreajuste a los datos de entrenamiento. Desde el punto de vista matemático los métodos de regularización suavizan (reducen la oscilación) de la función estimadora. Los límites de las regiones de decisión se suavizan, por lo que mejora la capacidad de generalización del modelo [19].

Dicha penalización puede consistir en reducir el número de *epochs* del entrenamiento, modificar dinámicamente la arquitectura de la red, o añadir un factor de penalización a la función de coste. Durante el desarrollo del clasificador se han empleado las técnicas de Dropout, Early Stopping y Regularización L1/L2.

**Early Stopping.** En la fase de entrenamiento el objetivo es minimizar la función de coste a lo largo de las iteraciones y epochs. El overfitting se produce porque el modelo comienza a diverger, como consecuencia de una sobreadaptación a los datos de entrenamiento. La función de coste de la partición de entrenamiento continúa decreciendo mientras que el coste en la partición de validación test pasa a tener una pauta creciente.

El tratamiento más sencillo consiste en detectar el momento en el que la efectividad del modelo comienza a degradarse y detener el entrenamiento en el mejor punto observado. Este método se conoce como Early Stopping, es sencillo de implementar



y evita el ajuste manual del número de epochs con el fin de obtener un resultado óptimo.

Durante la fase de experimentación de arquitecturas de redes neuronales se analizará la efectividad de esta técnica, de acuerdo a distintos valores de *paciencia*.

Para la implementación de Early Stopping se fija un valor de *paciencia* configurable. Este parámetro determina el número de iteraciones máximo seguidos en el que se permite que el modelo no experimente una mejora en el valor de coste. Es importante fijar un valor que no sea demasiado bajo, para que el modelo no sea sensible a subidas transitorias del coste, ni un valor demasiado alto para poder detener el entrenamiento en el momento adecuado.

**Dropout.** El método de *Dropout* [40] presenta un uso muy extendido para el tratamiento de overfitting en redes neuronales artificiales.

Esta técnica se aplica sobre las neuronas pertenecientes a las capas ocultas. Consiste en descartar aleatoriamente un porcentaje de neuronas por capa, en cada iteración, durante la fase de entrenamiento. Esta modificación fuerza la especialización de las neuronas vecinas, que tienen que adaptarse a la ausencia de aquellas descartadas. En el entrenamiento esto se traduce como una transformación en la distribución de pesos de la red, que presentará una tendencia menor hacia unas conexiones en concreto. El método de *Dropout* parte de la premisa de que esta adaptación mejora la capacidad de generalización del modelo.

Durante la fase de validación o testeo no se aplica descarte de neuronas, por lo que se trabaja con la arquitectura original, entrenada mediante esta técnica.

La idea tras el método de Dropout consiste en aplicar la media de predicciones de un número exponencial de modelos aprendidos que comparte parámetros [40]. Al aplicar dropout se está realizando una experimentación sobre un gran número de modelos distintos, sin necesidad de buscar la configuración adecuada para cada modelo. Esta técnica elimina temporalmente neuronas de la red, junto con sus conexiones asociadas. Los autores de esta técnica recomiendan utilizar por defecto una tasa de descarte de 0.50.

En cada iteración del entrenamiento, se descartan unidades en base a una variable que fija la probabilidad independiente de una unidad de ser descartada del modelo. Durante la fase de test se trabaja con la arquitectura original.

Dropout sigue la filosofía de Ensemble Learning, ya que la aplicación de la técnica puede ser vista como entrenar una colección de  $2^n$  submodelos distintos.

En tiempo de test no se aplica la técnica ya que no se considera factible basar la predicción en la media de las predicciones de todos los submodelos a generar.

Durante la fase de entrenamiento se emplea el método de gradiente descendiente, y se realiza una media de los gradientes sobre cada parámetro, en cada iteración. Los autores también destacan la posibilidad de aplicar en conjunto otras técnicas de regularización, como la regularización L2 o momentum, y recomiendan especialmente utilizar la regularización max-norm.

En el presente trabajo se analizan los beneficios de utilizar la técnica de dropout para mejorar la efectividad del clasificador ante el escenario de overfitting observado.

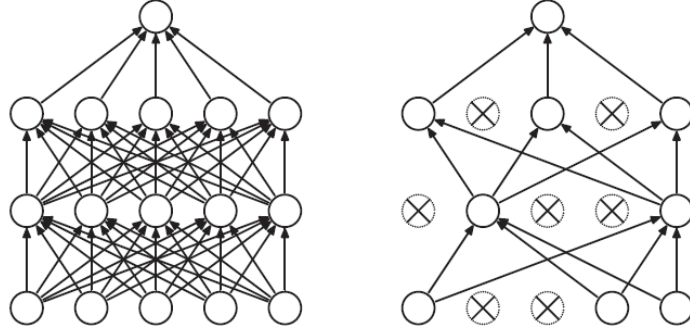


Figura 12: Red original (izquierda) y red resultante tras aplicar Dropout (derecha) [40]

Se tendrán en cuenta las observaciones expuestas a la hora de aplicar dicha técnica.

**Regularización L1 y L2.** Otra forma de regularizar el modelo es aplicar penalizaciones a lo largo del entrenamiento, con el fin de limitar hasta cierta cota la optimización a los datos de entrenamiento. Un regularizador añade una penalización al valor de coste, que condiciona el reajuste de pesos que se realiza al final del algoritmo de retropropagación. Existen varios regularizadores en función de los criterios para calcular el término de penalización, y al igual que en el caso de dropout, se aplican únicamente durante la fase de entrenamiento.

Los dos regularizadores más conocidos son el **regularizador L1 (LASSO)** y el **regularizador L2 (Ridge)** [20]. El regularizador de Lasso aplica el sumatorio de los pesos absolutos como penalización. En la práctica, es un selector de features, ya que minimiza gran parte de los pesos a cero, manteniendo aquellos que aportan más al modelo. Esto da lugar a modelos dispersos.

$$R(\theta) = E(\theta) + \alpha \sum_{i=1}^n |\theta_i|$$

Figura 13: Función de coste aplicando la regularización de Lasso (L1)

El regularizador de Ridge, por su parte, aplica una penalización en base a la suma del cuadrado de los pesos. En el entrenamiento esto da lugar a una reducción general en el valor de los pesos (sin llegar a suprimirlos como en Lasso), lo que se conoce como *weight decay*.

$$R(\theta) = E(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

Figura 14: Función de coste aplicando la regularización de Ridge (L2)

Entre la regularización L1 y L2, optamos por la L2, que suaviza los pesos de las conexiones, en lugar de L1, ya que se desea mantener todas las características utilizadas en el modelo.

### 2.4.3. Modelos de clasificación a partir de datasets desbalanceados. Técnicas correctoras

El uso de datasets desbalanceados es un problema frecuente en el área de clasificación supervisada. El origen de esta causa se debe a la dificultad de encontrar de forma natural ejemplos de ciertas clases, debido a que son casos poco observados o poco frecuentes. Los datos de entrada son un elemento muy valioso para el modelo de clasificación, ya que es la fuente en la que este fundamenta su aprendizaje. Datasets de entrada sesgados tienen tendencia a provocar que los clasificadores, de la misma forma, presenten un sesgo.

El modelo resultante puede llegar a ignorar una de las clases presentes, ya que el número de muestras representativas es demasiado pequeño para determinar con ellas una región de decisión. Si para una determinada clase no se disponen de muestras suficientes, estas se considerarán como ruido, dando lugar a situaciones de overfitting.

Para contrarrestar este efecto, se han definido diversas estrategias [1], englobadas en tres categorías:

- **Mecanismos *inbuilt*.** Consisten en técnicas que cambian los métodos de clasificación para introducir un *bias* (sesgo) hacia la clase minoritaria.
- **Métodos de preprocesamiento de textos.** Modifican la distribución de los datos para equilibrar la ocurrencia de muestras de las distintas clases. Existen dos opciones: realizar *oversampling* de la clase minoritaria, o realizar *undersampling* sobre la clase mayoritaria. Las técnicas de generación de muestras se pueden basar en el reemplazo de muestras, o en generación artificial de muestras [8]. La efectividad de cada alternativa se encuentra condicionada por el dataset considerado y el criterio utilizado para realizar el remuestreo.
- **Métodos de sensibilidad en las funciones de coste.** Consiste en asignar un coste mayor sobre aquellas clasificaciones erróneas realizadas relativas a la clase minoritaria.

Otro de los aspectos a tener en cuenta en el empleo de datasets desbalanceados es el uso de las **métricas** para evaluar la eficiencia del clasificador. Las métricas son el prisma a través del cual se evalúa el modelo resultante, pero dicha eficiencia medida se encuentra influida por la distribución de cada clase en los conjuntos de entrenamiento y validación/test. En estos escenarios resulta recomendable tener en cuenta métricas como la **curva ROC** (Receiver Operating Characteristic) y **AUC** (Area Under the Curve), que determinarán si el clasificador es óptimo [8]. También se propone el uso de multi-AUC y la media de la tasa de aciertos de cada clase [1].

Uno de los métodos más extendidos en el preprocesamiento de textos es el método **SMOTE** [8], que genera muestras sintéticas en base a muestras existentes en el

dataset de partida. Para ello aplica el algoritmo de k-vecinos, hallando la distancia entre dos muestras de la clase minoritaria y generando una muestra que se encuentre en un punto intermedio. Estas muestras sintéticas hacen aumentar las regiones de decisión, por lo que nos permite obtener clasificadores con una capacidad de generalización mayor. El empleo de SMOTE sobre la clase minoritaria introduce un bias sobre ésta, que disminuye la desventaja sobre el resto de clases.

La eficiencia de SMOTE ha sido validada en numerosos estudios. En el estudio original [8] se realiza una comparativa utilizando el algoritmo C4.5 (clasificador estadístico basado en un árbol de decisión) [46], Ripper (Sistema de aprendizaje por reglas)[10] y Red Bayesiana. Sobre estos clasificadores se utilizan nueve datasets distintos, en problemas de clasificación binaria. La mejor configuración que se cita en el artículo original consiste en combinar el sobremuestreo de la muestra mayoritaria con el inframuestreo de la minoritaria.

La técnica **SMOTE** se aplica principalmente sobre características continuas, aunque existen variantes como **SMOTE-NC** (SyntheticMinority Over-sampling TEchnique - Nominal Continuous), o **SMOTE-N**, que permiten el sobremuestreo de **características nominales y continuas** [8].

Las **clasificaciones multiclase** suponen un escenario más complejo con respecto a las clasificaciones binarias, ya que es necesario modificar cuidadosamente el dataset o la técnica de clasificación para no perjudicar la caracterización de alguna de las clases del modelo. El trabajo de Sáez, Krawczyk y Wozniak [1] ofrece un análisis detallado sobre la efectividad de técnicas de preprocesamiento como SMOTE, y aporta luz sobre la efectividad de esta técnica sobre modelos de clasificación multiclase. Sugieren realizar un análisis del dataset en busca de *outliers* o muestras poco frecuentes, que pueden ocasionar una superposición de las regiones de decisión de cada clase.

Las conclusiones de su estudio se encuentran basadas en la experimentación de 21 datasets multiclase, utilizando un clasificador C4.5 [46], un Support Vector Machine (SVM) y la regla de vecinos cercanos. A raíz del análisis concluyen que para la mayoría de los casos resulta preferible tratar mediante *oversampling* únicamente ciertas clases y tipos de ejemplos, ya que el *oversampling* de ciertas clases puede dar lugar a un deterioro del modelo. No obstante, en algunos datasets ha resultado más efectivo aplicar el tratamiento sobre todas las clases, por lo que se valida la necesidad de hacer un estudio más exhaustivo de las características de cada dataset.

Otra estrategia para paliar el efecto de datos de entrenamiento desbalanceados consiste en aplicar modificaciones en la fase de entrenamiento, utilizando el dataset original. Una de las opciones consiste en aplicar una **actualización de la tasa de aprendizaje** en base al *factor de atención* [12]. El factor de atención se calcula a partir de la proporción de muestras presentadas a la red neuronal durante el entrenamiento. El propósito de esta variable es el de aumentar la tasa de aprendizaje durante el procesamiento de una clase mayoritaria, y disminuirlo para el resto de clases.

Otros trabajos abogan por la combinación de varias estrategias, aplicando un **remuestreo** y modificando las **funciones de coste** [39]. Sobre la clase mayoritaria

se realiza una **clusterización**, y se toman los centroides como muestras representativas, descartando el resto. En la fase de entrenamiento se emplea una función de coste modificada que tiene como objetivo compensar la pérdida de información debida a la clusterización. En cuanto a la función de optimización, se proponen cuatro alternativas: gradiente descendiente, gradiente descendiente con momentum y learning rate variable, retro-propagación resiliente y la función de Levenberg-Marquardt. Los resultados son validados con una red neuronal hacia delante, y el clasificador resultante presenta una efectividad más igualada sobre las distintas clases.

Finalmente, otra de las opciones que se sugiere en el escenario multiclase se basa en **redefinir el problema** como una clasificación binaria, simplificando el problema y dando lugar a una distribución de clases posiblemente más equilibrada.

En este trabajo se ha optado por mantener el problema como una clasificación multiclase, ya que se considera que la información que aporta cada clase en este dominio resulta interesante. Para nuestro problema, resultaría de utilidad identificar únicamente qué noticias se encuentran relacionadas o no con un titular, pero se estaría desaprovechando la información completa que nos proporciona el dataset. El dataset distingue dentro de las noticias relacionadas noticias que apoyan, discuten o se encuentran en contra del titular, lo que aporta un matiz interesante.

Este proyecto estudia los efectos y posibles enfoques para lidiar con el desbalanceo de clases en escenarios multiclase. Se ha optado por utilizar la técnica SMOTE original para equilibrar la distribución de muestras de cada clase. El propósito del presente trabajo es evaluar la efectividad de esta técnica sobre la representación continua de características nominales (titulares y cuerpos de noticia), en concreto con modelos basados en bolsa de palabras y word2vec. La efectividad del clasificador resultante se ha valorado utilizando una red de neuronas multicapa y un clasificador de *Random Forest*.

Por otro lado, se han tenido en cuenta ideas a aplicar sobre el clasificador multicapa, como la variación dinámica de la tasa de aprendizaje. Se pretende validar si esta estrategia combinada de preprocesamiento del dataset y ajuste del clasificador consigue disminuir el error asociado a la distribución de clases que presenta el dataset.

## 2.5. Trabajos relacionados

En esta sección se describen los trabajos relacionados con el problema que afronta este trabajo. También se valoran las contribuciones que se aportan al campo de estudio en el que se enmarca.

Este trabajo se enmarca en la problemática relacionada con la **veracidad de la información** en los medios digitales, en la dificultad de aportar un procedimiento de gestión de calidad sobre las noticias que se publican y se comparten en Internet. Una forma de afrontar este problema es el de aportar un mecanismo automatizado para la clasificación de noticias, en base de su relación o no al texto, y más allá de eso, a su grado de conformidad con éste, como medida previa.

La tarea que aborda el presente trabajo consiste en analizar el posicionamien-

to de una noticia con respecto a un hecho. La **detección de posición** (*Stance Detection*) es una tarea relacionada con el **análisis de sentimientos** (minería de opinión), pero ambas tratan problemas distintos. El análisis de sentimientos determina si el texto aporta una opinión positiva, negativa o neutral con respecto a una entidad (película, situación, producto), mientras que el análisis de posición tiene como propósito detectar la disposición del autor (a favor, en contra, indiferente) con respecto a una entidad, por ejemplo, sobre las declaraciones de un personaje público o un tema de actualidad en concreto.

A la hora de dar solución a este tipo de problemas, el primer paso consiste por valorar qué técnicas se van a aplicar. Como trabajos previos se han consultado problemas de **posicionamiento de noticias**, algunos trabajos de **análisis de sentimiento**, y varios artículos de **clasificación de textos** en un sentido más amplio. Sobre el área de detección de posición se han estudiado las diversas soluciones aportadas para el problema definido en el desafío de **Fake News**, fase 1 <sup>1</sup> (el problema abordado por este trabajo) y la tarea de **SemEval 2016** [37], sobre la detección de posicionamiento de *Tweets* hacia personalidades o temas de interés. También se han consultado enfoques validados sobre datasets de NLP populares, como el dataset de críticas de *Yelp*<sup>2</sup> o el dataset de críticas de *IMDB*<sup>3</sup> para el análisis de sentimientos.

Las tres tareas principales que se abordan en la resolución de problemas de este tipo consisten en el **pre-procesamiento de datos de entrada**, **generación de representaciones** de los textos de entrada, y diseño del **modelo clasificador de textos**. A continuación se describen los enfoques observados.

### 2.5.1. Detección de posición. Técnicas de clasificación de textos.

Los trabajos en el área de posicionamiento o clasificación de textos ofrecen múltiples enfoques para afrontar problemas como el tratado en este trabajo. En base a la literatura existente, se pueden concluir varias estrategias efectivas. La primera es el uso de vectores de *embedding* para la construcción de representaciones, y cualquier otra característica de apoyo que aporte información sobre el target o el texto. La segunda hace alusión a los modelos a utilizar, en la que podemos tomar dos estrategias, utilizar **clasificadores clásicos** como cadenas de Markov o máquinas de vectores de soporte (SVM), o **arquitecturas de red neuronal**, como redes neuronales recurrentes o redes recurrentes LSTM.

**Enfoques basados en modelos clásicos.** Uno de las alternativas dentro de este área consiste en un sistema inteligente basado en **modelos de markov (HMM)** y **Máquinas de soporte de vectores (SVM)** para la categorización de noticias en base a su temática [28]. Las noticias se clasifican en base a un conjunto de categorías predefinidas. Con el **modelo de Markov** se extraen las **palabras claves** del texto. Cada texto se caracteriza como una secuencia de observaciones, y cada observación

---

<sup>1</sup><http://www.fakenewschallenge.org/>

<sup>2</sup><https://www.yelp.com/dataset>

<sup>3</sup><http://ai.stanford.edu/~amaas/data/sentiment/>



es un vector de características. Las observaciones son tomadas como entradas del modelo y los estados objetivo de la cadena representan las etiquetas de categorización del texto (palabras clave). Las palabras clave de un texto se toman como entrada del **clasificador SVM multiclase**, formado por 3 clasificadores binarios, uno por cada clase considerada. Cada clasificador se entrena con muestras positivas y negativas con respecto a la clase.

La tasa de aciertos por clase se encuentra en el rango de 90 % al 96 %, aunque se indica que en algunos casos la selección de palabras clave proporciona cierta ambigüedad y no es determinante de la temática en algunos casos. No obstante, el uso de modelos de Markov constituye una mejora en la tasa de aciertos con respecto a utilizar únicamente SVM.

Otros trabajos afrontan el problema de análisis de sentimientos y detección de posición como uno solo, con el propósito de determinar si características de un problema podrían beneficiar al otro. El caso de uso consiste en la clasificación de Tweets pertenecientes al dataset de la tarea de SemEval del año 2016.

La arquitectura consiste en un **clasificador SVM** que toma como entrada estructuras de formato **n-grama**, características asociadas al sentimiento (extraídas a partir de un glosario o *lexicon*) y vectores de medias de **embedding** para los datos sin etiquetar. Construyen un modelo por cada uno de los temas considerados (cinco en total), que ofrece un resultado mejor con respecto a utilizar uno solo. El modelo de *Word2Vec* fue entrenado con el corpus de dominio del problema [56].

Esta estrategia de acción conjunta concluye que las características de sentimiento no son relevantes para alimentar el modelo de clasificación de la posición. En el apartado de representaciones ratifican la mejora que supone utilizar vectores de embeddings en el proceso de generar representaciones de textos.

La estrategia consistente en utilizar el algoritmo de SVM en conjunción con HMM parece una configuración habitual, y algunos artículos se centran en ofrecer una comparativa de **modelos generativos** versus **modelos discriminativos** dentro de esta corriente. Los modelos generativos tienen en cuenta la unión de probabilidades entre la variable observada y la objetiva, mientras que los modelos discriminativos se basan en la probabilidad de la variable objetivo en base a la variable observada.

El problema abordado en el trabajo de Hasan y Ng [22] consiste en la clasificación de posición de debates ideológicos, y en él abordan cuatro dominios distintos. Para examinar de qué manera impacta la complejidad del modelo en la tarea de detección de posición analizan tres grupos de modelos. El primero consiste en un **clasificador binario de redes bayesianas** (modelo generativo) y otro discriminativo basado en **máquinas de vectores de soporte**. El segundo grupo representa los modelos de secuencia, modelos toman como entrada el post y generan una salida de etiquetas. Como etiquetadores de secuencia emplean un modelo generativo basado en **modelos de Markov (HMM)** y otro discriminativo de **CRF (linear-chain Conditional Random Fields)** [29]. El tercer grupo de modelos son modelos de grano fino, que determinan la posición de la publicación en general y las posiciones de cada una de las frases que lo conforman, para lo que optan por modelos de **redes bayesianas** y modelos de **redes de Markov**.

Las características obtenidas a partir del corpus de entrada son unigramas y bigramas, como vectores one-hot, a lo que incorporan estadísticas sobre el documento, dependencias sintácticas y el conjunto de características extraídas del *post* anterior. A raíz de los estudios no queda concluyente cual de los dos modelos (discriminativo o generativo) es más efectivo, pero se establece que los modelos de secuencia, en este caso los modelos de Markov, son más efectivos.

Una de las soluciones aportadas al desafío de *Fake News* apuesta por el uso de clasificadores clásicos [6]. En esta alternativa, la representación se construye a partir de n-gramas que aparecen tanto en el titular como en el artículo, y que tienen una cierta relevancia. Este procesamiento previo les permite predecir de entrada las muestras de noticias relacionadas con respecto a las clases no relacionadas, alcanzando una tasa de aciertos del 61 %. Utilizan un clasificador logístico, entrenado únicamente con los titulares, para detectar la posición de las noticias. Después aplican una cota de confianza, para determinar si el resultado más probable devuelto por el clasificador es apto o no. Si no se supera dicha cota, se predice la clase usando tres clasificadores binarios de *de acuerdo - en contra*, *de acuerdo - discute y discute* - *en desacuerdo*.

**Enfoques basados en redes neuronales artificiales.** Se pueden encontrar una amplia variedad de trabajos que aplican técnicas de deep learning para clasificación de textos. El artículo *Fine-grained opinion mining with RNN and Word Embeddings* [33], propone generar representaciones del texto mediante word embeddings, y utilizar como clasificador un modelo de red neuronal, para lo que exponen tres arquitecturas: las arquitecturas clásicas de **Elman** [16] y **Jordan** [26], y el **modelo de Long-Short Memory(LSTM)** [24]. Su estudio ratifica el potencial de word embeddings, en su uso combinado con RNN, sobre otras técnicas como los modelos CRF. Para mejorar los resultados de la arquitectura expuesta, se propone la **inclusión de características lingüísticas simples**. Para capturar relaciones de los términos con respecto términos anteriores y posteriores dentro de la misma frase proponen utilizar **redes recurrentes bidireccionales** [52], aunque en el análisis reportado no ofrecen una mejora con respecto al uso de redes recurrentes unidireccionales.

Dentro del grupo de modelos RNN, muchas experimentaciones optan por las **redes bidireccionales** [52], en las que se produce una realimentación de izquierda a derecha y de derecha a izquierda. Una de las soluciones al desafío SemEval 2016 propone el uso de **LSTM** para construir la representación del target y otra red LSTM posterior para codificar el texto utilizando como estado inicial la representación del target. El vector de salida se emplea para predecir la posición del tweet con respecto al target. Otra arquitectura propuesta se fundamenta en el uso de encoding bidireccional condicional, partiendo del primer enfoque. Se utiliza una representación de dos vectores por tweet y target, uno calculado procesando los elementos de izquierda-derecha, y otro de derecha a izquierda. Los word embeddings se inician usando un modelo pre-entrenado de vectores de word2vec [2].

La arquitectura de LSTM constituye una mejora con respecto a las redes re-



corrientes clásicas, que tiene como objeto paliar el conocido problema de *Exploding Gradient* [43]. Una de las soluciones al desafío SemEval 2016 se basa en un modelo formado por dos redes recurrentes LSTM. El primer modelo recurrente toma como entrada vectores one-hot, y la primera capa del sistema es una capa de embedding, que genera la representación densa de los términos. Este primer modelo se entrena para predecir hashtags (temas) de manera no supervisada, y se utiliza en la segunda red, que toma como entrada los tweets y se inicializa con las predicciones de los temas [69].

También se ofrecen alternativas muy interesantes, que utilizan las redes recurrentes para implementar el concepto de **atención en el aprendizaje**. El artículo *Stance Classification with target-specific neural attention networks* propone una arquitectura de red conocida como **Target-specific attentional network (TAN)** [14], que combina un modelo de red recurrente con una red LSTM y un extractor de atención con respecto al target / entidad objetivo. La red es utilizada para obtener las partes más importantes del texto que estén altamente relacionadas con el target. La representación está basada en un embedding por target y otro por texto. El clasificador está compuesto por dos modelos: un **modelo de red neuronal hacia delante** que tiene como propósito determinar el factor de atención del modelo, y un modelo de red **LSTM bidireccional**, encargada de extraer características del texto. El uso de la red de atención mejora perceptiblemente la capacidad de clasificación del modelo final, ya que el modelo es capaz de extraer las partes importantes del texto.

Dos de las soluciones ofrecidas al desafío de Fake News utilizan el concepto de **factor de atención** [47] [70]. El primero utiliza arquitecturas de **red recurrente** y **LSTM**. Proponen una arquitectura basada en redes neuronales hacia delante, utilizando como representaciones vectores de medias de embedding. La segunda arquitectura que proponen se basa en el uso de dos redes LSTM, una para los titulares y otra para los cuerpos de noticia, para generar las representaciones de cada texto. Otra opción que proponen consiste en un modelo de **codificación condicionada**, enlazando las dos arquitecturas de LSTM, para que la representación del titular influya en la representación del cuerpo de noticia. También se propone una **variante bidireccional (Bidirectional Conditional Encoding, BCE)**. Finalmente implementan una versión modificada del modelo de **Attentive Reader Model (ARM)**[23]. De todas las arquitecturas expuestas, las que ofrecen mejores resultados son el modelo de red neuronal hacia delante, y el modelo de Attentive Reader. No obstante, apuntan un escenario de overfitting en este último caso, y dificultad para detectar ciertas muestras de clases que apoyan, discuten o contradicen la noticia. El segundo de los trabajos citados experimenta con ideas similares, empleando un clasificador recurrente bidireccional con factor de atención [70].

Se han diseñado arquitecturas específicas para problemas de minería de opinión, sobre textos de larga longitud. El modelo propuesto es una modificación del modelo de LSTM, y recibe el nombre de **SR-LSTM (LSTM with sentence representations)** [48]. La primera capa aprende sobre la semántica del texto aplicando LSTM, tomando como entrada vectores de embedding, y genera un vector por frase. Estos

vectores son la entrada del segundo modelo de LSTM, que se encarga de generar representaciones del texto en su conjunto. Estas representaciones son las que se toman como entrada del modelo clasificador final. Ofrece mejores resultados en comparación con otros modelos de RNN considerados para el estudio, entre los que se encuentran redes LSTM bidireccionales, redes con **células GRU** [9] o RNN clásicas. Esta alternativa también es apta para longitudes de texto menor, pero es necesario hacer un reajuste de los hiperparámetros.

En el artículo *Deep learning for sentiment analysis of movie reviews* [45], se aporta una comparativa de métodos clásicos como Random Forest, SVM (Support Vector Machines) y regresión logística con respecto métodos basados en redes recurrentes. El problema abordado consiste en detectar el sentimiento de críticas de cine del portal IMDB.

Las técnicas consideradas consisten en el modelo de Bolsa de palabras y representaciones de vectores embebidos, basados en Word2Vec. Para la clasificación, los autores realizan una experimentación basada en redes recursivas (**RNTN, Recursive Neural Tensor Networks**). Concretamente se utiliza una variante conocida como *Low Rank*[4], que limita las dimensiones del vector de pesos de la red. Esta configuración provoca una reducción en el cómputo por iteración. Como optimizador se utiliza el Gradiente *adaGrad* [15], aplicando una regularización L2. En la experimentación con redes neuronales recursivas se obtienen unos tiempos de convergencia altos, utilizando la variante RNTN estándar. Para la variante de *Low Rank* se obtienen unos tiempos de convergencia más razonables.

El potencial de reutilización de las arquitecturas de red para otros problemas tiene su aplicación en problemas de aprendizaje multitarea. El **aprendizaje multitarea (multi-task learning)** consiste en estudiar las correlaciones entre cada tarea y realizar una clasificación en paralelo de las mismas. Un enfoque común en este tipo de problemas consiste en utilizar las mismas capas inferiores para los distintos modelos. Las capas posteriores son específicas de cada problema. Se proponen tres modelos de red diferentes [34]. El primero consiste en que todas las tareas compartan la primera capa de la red LSTM y una capa de embedding, la segunda, en que cada tarea tenga su propia capa de LSTM, y la tercera, asigna a cada tarea una capa e introduce una capa bidireccional que comparten todas las tareas, para compartir conocimiento a través de las distintas tareas. Las redes se validan con distintos datasets de análisis de sentimientos sobre críticas de películas. Se utilizan embeddings de word2vec para generar la representación densa de los textos. Los resultados constatan mejoras de las tres arquitecturas con respecto los resultados base, y el modelo que se comporta mejor es el tercero. Los autores apuntan que la arquitectura utilizada no presenta un coste computacional extra con respecto a la arquitectura de LSTM clásica, ya que logra una convergencia mejor con respecto a esta última. Se reporta un análisis sobre los tipos de error en la clasificación, que principalmente se deben a muestras con una estructura de frase compleja y textos que contienen frases de sentido metafórico.

El uso de **autoencoders** no ha sido una tendencia predominante en los trabajos de clasificación de textos analizados. Se trata de una arquitectura muy conocida

dentro de RNN, que se pueden emplear para generar representaciones de manera no supervisada. No obstante, no ha sido una tendencia predominante en los trabajos relacionados de clasificación de posición. Una de las soluciones al desafío de SemEval 2016, expone el uso de un autoencoder que toma como entrada un modelo de bolsa de palabras de 50000 términos, para construir las representaciones de entrada de los textos del modelo. [3] El modelo clasificador consisten en un clasificador logístico de tres clases, optimizado con regularización L2. El autoencoder es un tipo de red neuronal recurrente que permite generar representaciones de manera no supervisada, en este caso genera representaciones de 100 dimensiones. En el trabajo plantean varias configuraciones en las que se aplica el autoencoder únicamente sobre el tweet, sobre el tweet y el target, y sobre ésta última, se prueba a generar una representación por elemento o una representación combinada. El la mejor configuración reportada es aquella en la que se utiliza el autoencoder para generar las dos representaciones, obteniendo un *f1-score* de 0.37.

Existe otra variante interesante que aboga por el uso de **redes convolucionales** [68]. Este enfoque analiza el uso de estas redes como un **sistema de votos** para la predicción, similar a los modelos de ensemble. Al igual que otros casos, se emplean vectores de embedding, haciendo uso del modelo de word2vec pre-entrenado oficial. La red convolucional está formada por una capa convolucional, una capa de pooling y una capa softmax. El sistema de votos se construye a partir de los resultados de la capa de softmax. La capa convolucional tiene como propósito extraer patrones de los textos. La capa de pooling simplifica la información procesada por la capa convolucional, generando un vector de menos dimensiones, y se ha configurado con la función de maxpooling (se genera la salida a partir del valor máximo de cada columna).

En el artículo *Convolutional neural networks for sentence classification* [27], se presenta otra solución basada redes neuronales convolucionales para la tarea de clasificación de textos. Se propone una arquitectura convolucional que toma como entrada las representaciones de los términos, una capa de pooling (variante max-over time), y una capa permanentemente conectada con dropouty regularización l2 y función de *softmax* a la salida. Toma frases formadas por vectores de longitud fija. Proponen varias estrategias para inicializar los vectores asociados a términos: aleatorio, embeddings de word2vec, y una variante multicanal, con dos conjuntos de vectores. El filtro convolucional se aplica sobre los dos canales, pero el gradiente únicamente se propaga sobre uno de los canales. No se constata una mejora de la variante multicanal sobre el resto, pero sí que se establece que inicializar los vectores usando word2vec ofrece mejores resultados con respecto la inicialización aleatoria.

Otros trabajos proponen **arquitecturas híbridas** de redes de neuronas, combinando **redes neuronales convolucionales** y **redes neuronales recurrentes** para clasificación de textos. Uno de los sistemas propuestos emplea la red convolucional para extraer una secuencia representativa del texto de entrada, que se emplea como entrada del modelo de LSTM. La arquitectura propuesta es capaz de capturar características locales en secuencias cortas (*phrases*) y la semántica de la frase a la que pertenecen. El problema de clasificación abordado consiste en una clasificación

de sentimientos en 5 clases y una clasificación de preguntas en base a un conjunto de 6 categorías. [71]

La red convolucional se utiliza para detectar relaciones temporales o espaciales de los textos, mientras que la red recurrente establece correlaciones del texto como secuencia, capturando relaciones de largo alcance. Esta primera red toma como entrada los vectores de características del texto, para generar una representación de alto nivel mediante estructuras de n-tuplas, que representan subconjuntos dentro de la frase. Estas n-tuplas se toman como la entrada secuencial del modelo recurrente. La salida del estado oculto en el último estado es la representación del texto. Para poder conectar las dos arquitecturas se elimina la capa de pooling final, por lo que la red convolucional únicamente aplica la operación de convolución.

Otro ejemplo de arquitectura híbrida utilizan una **red recurrente bidireccional** para capturar la semántica de los textos y crear la estructura de representación [30]. Esta red se considera como la capa de convolución de la estructura híbrida, y la siguiente capa consiste en una **función de pooling**. La capa de pooling convierte los vectores de longitud variable a longitud fija. La capa de salida aplica una función de softmax. Aplican la arquitectura sobre datasets de referencia sobre análisis de sentimientos y categorización de temática. Establecen que las arquitecturas de CNN son mejores a la hora de capturar características semánticas, ya que el modelo de RNN se encuentra sesgado por la la secuencialidad del texto y la memoria que contempla (pierde la imagen de conjunto). La arquitectura híbrida ofrece mejores resultados con respecto el uso de CNN o RNN por separado.

Para el área de clasificación de textos también se han valorado clasificadores que aplican *Ensemble Learning* [64] [?]. El clasificador está formado por dos capas, la primera formada por una serie de clasificadores, y una segunda que se encarga de obtener la predicción final tomando como base las distintas predicciones de los clasificadores. En total se definen 5 clasificadores secundarios, siendo cuatro de ellos modelos de **MLP (perceptrón multicapa)** y otro un clasificador logístico de *one-vs-all*. El clasificador de la última capa es un árbol de decisión que utiliza como características la salida producida por cada clasificador. El modelo de ensemble supone una mejora del 1.6 % con respecto a los clasificadores individuales.

Finalmente, existen trabajos que optan por reinventar las arquitecturas básicas más extendidas, en lugar de crear arquitecturas híbridas, proponiendo nuevas arquitecturas de modelos. Se propone una arquitectura de deep learning conocida como **DAN2 (Dinamic ANN)** [18]. Consiste en una arquitectura dinámica para redes neuronales. En esta arquitectura, el número de capas ocultas no está fijado a priori. Las capas se van generando dinámicamente hasta que se alcanza cierto nivel en la tasa de aciertos. Cada capa oculta está formada por 4 nodos. Un nodo de la arquitectura se encarga de almacenar lo que refieren como el conocimiento acumulado del modelo en la iteración anterior, otro alberga el elemento residual a lo largo del entrenamiento, otro el elemento residual actual, y otro el bias, fijado a 1. Los nodos aportan una representación del proceso de aprendizaje iterativo. Defienden que esta arquitectura resulta más efectiva que RNN.

### 2.5.2. Discusión de técnicas y aportación del trabajo.

Los trabajos relacionados aportan una amplia panorámica de las tendencias de estudio en el área de detección de posicionamiento de noticias. En varios de los trabajos referidos se apunta que la mayor complejidad de estos problemas es la **detección de entidades implícitas** [56] [2]. Es decir, identificar que una posición positiva o negativa con respecto a una personalidad implica una posición contraria con respecto a otra personalidad no mencionada. Esta afirmación apunta la necesidad de abordar las representaciones en un contexto más amplio. Para este problema es interesante emplear vectores de embedding, ya que nos proporcionan un espacio continuo que representa las relaciones semánticas de los términos. Dos entidades con significados diversos tendrán representaciones alejadas en el espacio, y le pueden servir al clasificador para caracterizar mejor las distintas clases. Esta técnica de representación se valida en gran parte de los artículos referenciados, por lo que es una de las técnicas principales que formarán parte del clasificador de este trabajo.

En cuanto a la similaridad de esta categoría de problemas con respecto al análisis de sentimientos, se han encontrado evidencias para considerar estos problemas de forma separada. La estrategia de acción conjunta propuesta en *Detecting stance in tweets and analyzing its interaction with sentiment* [56] concluye que las características de sentimiento no son relevantes para alimentar el modelo para clasificación de la posición.

En base a la literatura revisada, se observa una tendencia de un mejor comportamiento por parte de los modelos neuronales frente a los enfoques clásicos. El uso de **arquitecturas recurrentes** y sus múltiples derivadas ha cobrado una amplia aceptación en problemas de lenguaje natural, y ofrece una perspectiva muy interesante y prometedora. Uno de los puntos fuertes de estas arquitecturas es que permiten entradas de distinta longitud. Por contra, la arquitectura es más compleja conceptualmente, con respecto a otras como la arquitectura de red hacia delante, lo que puede provocar tiempos de convergencia mayores.

A pesar del potencial de las arquitecturas recurrentes, resulta de utilidad tener en cuenta otras arquitecturas que han demostrado ser efectivas en sub tareas concretas, como es el caso de la construcción de representaciones de vectores de *embedding* (Word2Vec), que se basa en redes neuronales de dos capas optimizada convenientemente [36]. Es necesario seguir preguntándose si para determinados problemas de clasificación de texto existen otras alternativas competentes de forma similar.

Varios de los trabajos referidos aportaban soluciones al desafío de *Fake News*, algunos ofrecían soluciones apostando por redes neuronales recurrentes [47] [70], redes convolucionales, *ensemble de redes multicapa* o clasificadores clásicos [6].

Lo que este trabajo ofrece es el estudio de este problema desde otro enfoque, utilizando **redes neuronales hacia delante** y **representaciones embebidas**. La aportación al conocimiento que ofrece este trabajo es justificar si este tipo de arquitectura está a la altura para este tipo de problemas. El propósito de este trabajo es el de proporcionar un análisis detallado sobre el uso de redes neuronales hacia delante para la clasificación de noticias, así como las fortalezas y debilidades a raíz de su uso.

Para establecer una comparativa se han planteado distintos escenarios, que implican el uso del modelo clásico de bolsa de términos o de vectores de *embeddings*, en el caso de las representaciones, y el modelo de *Random Forest* o el modelo de red neuronal como modelos de aprendizaje. Durante el estudio e implementación deberán afrontarse cuestiones de diseño comunes en problemas de Machine Learning, como la distribución de los datos y su impacto en el aprendizaje, el ajuste del modelo a los mismos y el tiempo de ejecución versus la efectividad del modelo.

Finalmente, otra aportación del trabajo consiste en ofrecer el clasificador al usuario final. En este trabajo se propone un **servicio web** que expone a los usuarios el potencial del clasificador. El propósito es el de proporcionar al usuario una herramienta para realizar una categorización de la noticia y exponer un caso de uso en el que el clasificador tenga cabida. Este trabajo documenta el proceso de diseño, desarrollo y refinamiento del modelo clasificador y la plataforma web de acceso al servicio.





### 3. Evaluación de riesgos

En esta sección se presenta el problema de clasificación abordado, y los recursos que se utilizarán para su implementación. Se describirá el dataset de entrada escogido y la arquitectura de plataforma del filtrador inteligente de noticias.

#### 3.1. Caracterización del problema

La sociedad actual se encuentra permanentemente conectada. Tiene en su poder múltiples plataformas en las que informarse de las noticias de su entorno y discutir las con sus allegados. Sin embargo, las fuentes desde las cuales se obtiene información (redes sociales, portales de agregación de noticias, etc.) presentan mucho ruido, ya que en muchas ocasiones nos llegan noticias sesgadas en función del medio o de nuestro círculo cercano, mientras que en otras accedemos inconscientemente a noticias falsas o basadas en hechos poco fundamentados. Puede llegar a ser tan peligroso estar desinformado como estar mal informado, y la percepción que tenemos del mundo influye en nuestras decisiones y en la credibilidad que le damos a entidades o personas.

El concepto de **posverdad** (distorsión deliberada de la realidad) se encuentra más vigente que nunca, y plataformas como Facebook han comenzado a incluir como parte de su estrategia principal la mitigación de la difusión de noticias falsas. Existe la necesidad de investigar y desarrollar mecanismos de defensa que nos permitan cribar e identificar la veracidad de la información.

La mayor problemática de esta situación es que discernir la veracidad de un hecho es un proceso muy complejo en el que intervienen múltiples factores. Una noticia tiene que ser investigada con el fin de encontrar hechos de la realidad que la refuten o la acepten. Resultaría muy provechoso disponer de un mecanismo automático para clasificar noticias, pero probablemente no sería posible establecer tan sólo con un análisis del texto si una noticia es cierta. Lo que sí resulta abordable es analizar los hechos y datos que se exponen en la noticia y establecer si existe alguna contradicción o algún tipo de sesgo en la redacción de ésta. Este mecanismo supondría un punto de entrada para aquellos que deseen investigar o contrastar una noticia.

Nos encontramos ante un problema de detección de posición (Stance Detection). Se trata de una clasificación supervisada, en el que la entrada del sistema será un titular y su correspondiente cuerpo de noticia, perteneciente a la misma noticia o a noticias distintas. El clasificador a desarrollar es multiclase y las etiquetas son excluyentes entre sí. La clasificación de salida se encuentra entre las siguientes:

- **De acuerdo.** El texto apoya el hecho que expone el titular.
- **En contra.** El texto refuta el titular.
- **Discute.** El cuerpo de la noticia discute el mismo tema que el titular, pero no toma ninguna posición al respecto.



- **No relacionado.** El cuerpo de la noticia y el titular hacen referencia a temas distintos.

Este trabajo toma como base el desafío *Fake News Challenge* <sup>4</sup> del año 2017 [66], lanzado por un grupo de 100 voluntarios y diversos equipos del campo académico y de la industria de noticias. El desafío parte y pretende ser la continuación del dataset *Emergent* [17], que contiene noticias para el análisis de posicionamiento, obtenidas de distintos portales web de rumores y redes sociales.

El problema descrito forma parte del campo de periodismo computacional, que consiste en la aplicación de técnicas de computación a tareas como la recolección de información, organización y distribución de noticias. Las competencias principales de esta corriente son la curación de contenidos, comunicación y búsqueda y verificación de información.

La motivación de utilizar las guías del desafío es el de disponer de un dataset fiable para investigar sobre el potencial de las técnicas de Deep Learning en problemas de procesamiento de lenguaje natural. El propósito de este trabajo es estudiar el papel de las redes neuronales artificiales en el campo clasificación de noticias. Para ello se utiliza el framework de TensorFlow [59], y se estudian sistemas de representación basados en conteo (bolsa de palabras) y representaciones basadas en redes de neuronas (Word2Vec, redes neuronales recurrentes). La representación más eficaz conseguida se ha empleado como entrada del modelo de Machine Learning que realiza la clasificación. En base a dicho sistema se ha desarrollado un filtrador inteligente web de propósito general para explorar las noticias.

### 3.2. Dataset para la clasificación de noticias.

El dataset utilizado para la clasificación surge a partir de un trabajo de *crowd-sourcing* de especialistas del área de periodismo [17]. El corpus de muestras original disponía de tres clases, de acuerdo al posicionamiento de la noticia respecto al titular: *a favor*, *en contra* y *discute*. Para ello, se han empleado noticias de ámbito global, estadounidense y otras de temática tecnológica, sumando un total de 300 titulares y 2595 noticias relacionadas.

Para analizar el potencial del dataset los autores implementaron un regresor logístico con regularización L1. Como características de entrada utilizaron una representación de Bolsa de términos del titular, y una serie de características adicionales basadas en la relación de términos entre la noticia y titular. También se utilizó una representación basada en Word2Vec, obteniendo el vector de representación de la noticia multiplicando los vectores de términos, y empleando el modelo pre-entrenado oficial. La mejor configuración reportada consigue una tasa de aciertos en torno al 73 %.

El dataset *Emergent* puede ser utilizado para clasificación de posicionamiento de noticias, como parte del proceso de contrastación de hechos, o para crear un sistema inteligente para el resumen de noticias.

---

<sup>4</sup><http://www.fakenewschallenge.org/>

En este proyecto se emplea una versión posterior y ampliada, con un mayor número de muestras y una nueva clase, que hace referencia a noticias que no están relacionadas con su titular.

### 3.3. Filtrador web inteligente de noticias.

La herramienta web desarrollada recibe el nombre de *Stance Detector*<sup>5</sup> y permitirá que los usuarios realicen consultas en lenguaje natural sobre el posicionamiento de una noticia con respecto a un titular. La herramienta está concebida para ser utilizada por usuarios finales, consumidores de fuentes digitales de información, como periódicos online o redes sociales.

El filtrador web planteado sigue una arquitectura cliente-servidor. La aplicación se encuentra formada por un cliente web ligero al que accederán los usuarios, y el extremo servidor, que dispone de la lógica necesaria para acceder al modelo y obtener la clasificación de la noticia.

El extremo servidor será el encargado de tratar la consulta, y para lo cual generará una representación vectorizada del texto, utilizada como entrada al modelo. El modelo consiste en un clasificador multiclase basado en redes neuronales. Se trata de un clasificador logístico, que genera una salida que indica la probabilidad de pertenencia a cada una de las clases identificadas por el modelo, de la que finalmente se reporta la solución con mayor probabilidad. El esquema de entrenamiento considerado es de tipo *batch*. El entrenamiento del modelo se realiza una vez, a partir del dataset de entrenamiento utilizado. Una vez entrenado, se guarda el modelo en el servidor para su posterior consulta.

Se ha optado por un mecanismo de entrenamiento batch, ya que se desea partir de una fuente de datos fiable y estática. No se ha considerado ningún mecanismo de realimentación del modelo. Realizar un entrenamiento iterativo del modelo sería una alternativa interesante, pero también plantea una serie de problemas como la fiabilidad de los datos de entrada o monitorización de la calidad de las diversas versiones del modelo. Para este trabajo se ha optado por concentrar los esfuerzos en la obtención de un modelo fiable a partir de los datos de entrada, de los que se conoce su fuente y se encuentran caracterizados.

Los usuarios accederán a la plataforma web para obtener la clasificación de las noticias, y podrán visualizar la noticia con la clasificación ofrecida por el modelo de red neuronal. Los componentes lógicos del sistema son los siguientes:

- **Interfaz de acceso web a la plataforma.** Interfaz web cliente accesible a los usuarios. Es el punto de entrada de interacción con la plataforma.
- **Enrutador de peticiones.** Consiste en una API REST que expone la funcionalidad del clasificador al extremo cliente. Toma las peticiones del cliente y es el encargado de trasladar la consulta a las capas inferiores del servicio. Tras la obtención del resultado genera una respuesta interpretable por el cliente web o gestiona los posibles problemas de acceso al servicio.

---

<sup>5</sup><https://ailopera.github.io/stanceDetector.github.io/>

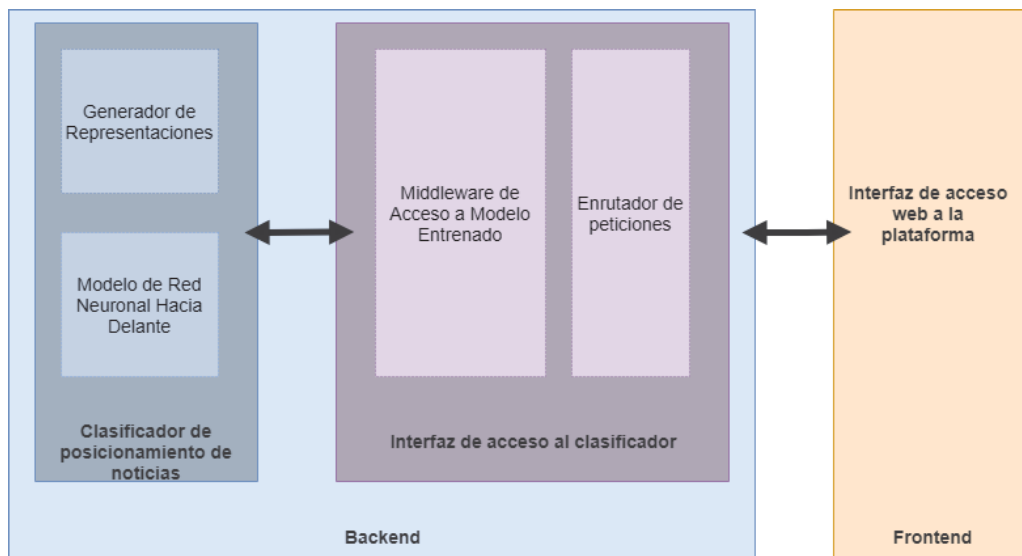


Figura 15: Diagrama lógico del filtrador web de noticias

- **Middleware de acceso al modelo entrenado.** Este módulo realiza la carga del modelo neuronal ya entrenado en memoria, para realizar consultas sobre éste. Este mecanismo intermediario se apoya en el módulo generador de representaciones obtener la representación. Dicha representación será empleada para realizar la consulta al modelo. Una vez obtenida la respuesta, genera una estructura intermedia de los resultados, que devolverá al mecanismo enrutador, que compone la respuesta final.
- **Generador de Representaciones.** Se encarga de construir una representación de vector de características, desde lenguaje natural, para realizar consultas al modelo o para generar las características de entrada del modelo en tiempo de entrenamiento.
- **Modelo de Red Neuronal.** Consiste en un clasificador configurable que realiza el entrenamiento *offline* del modelo y almacena en el sistema una versión estable y entrenada del clasificador.

El desarrollo del sistema se divide en dos fases: **desarrollo del módulo clasificador** y **desarrollo del entorno de producción** del filtrador web. La primera fase consiste en una experimentación que tiene como propósito determinar de forma cuantitativa qué estrategias de representación y entrenamiento resultan más provechosas.

Se han explorado dos generadores de representaciones distintos: uno basado en modelos de bolsa de palabras y otro en vectores embebidos de Word2Vec. Para el modelo de red neuronal hacia delante se han considerado diversas configuraciones de red neuronal.

Una vez terminada la experimentación se obtiene una panorámica de las diversas opciones, y se escoge aquella configuración que ofrezca una caracterización de

clases más precisa. El mayor desafío de esta clase es determinar las configuraciones específicas que se estudiarán, y valorar y exponer los puntos fuertes y debilidades de cada opción.

La segunda fase consiste en la puesta en marcha del entorno de producción. Una vez obtenida una versión estable del clasificador de posicionamiento de noticias, se ha implementado la interfaz de acceso al servidor, y la interfaz web cliente. El servicio expone el clasificador a los usuarios y los requisitos que debe cumplir son los siguientes:

- **Alta tasa de aciertos y caracterización precisa de las clases.** La herramienta deberá proporcionar una alta fiabilidad a los usuarios, para que lo consideren como parte de su experiencia de consumo de noticias en red. En la fase de experimentación de modelos se apostará por aquellas configuraciones o técnicas que nos permitan mejorar la calidad del clasificador, en las tres competencias básicas (*accuracy*, *precision* y *recall*).
- **Bajo tiempo de respuesta.** El tiempo de respuesta del servicio es clave para permitir a los usuarios acceder a las noticias en tiempo real. Se valorarán decisiones de diseño para reducir los tiempos de respuesta en la medida de lo posible.
- **Interfaz de uso sencilla.** El propósito del sistema es proporcionar una herramienta intuitiva. El sistema estará formado por dos vistas, la primera consistente en un formulario web, y la segunda en un visualizador de la noticia junto con la clasificación obtenida de la noticia.

El proceso de tratamiento de noticias desde que entra en el sistema hasta que se procesa es el siguiente. Se toman el titulares y el cuerpo de noticia asociado, y se les aplica un preprocesamiento con el propósito de eliminar secuencias no deseadas, y estandarizar las diversas ocurrencias de un término dado. El siguiente paso consiste en generar una representación de cada elemento en vectores densos. Dichos vectores alimentan un modelo de clasificador multiclase entrenado para la clasificación. El modelo genera una salida en base a la predicción realizada. El sistema tomará la predicción y compondrá la respuesta, que se utilizará en el lado cliente para componer la respuesta que verá el usuario en la plataforma.

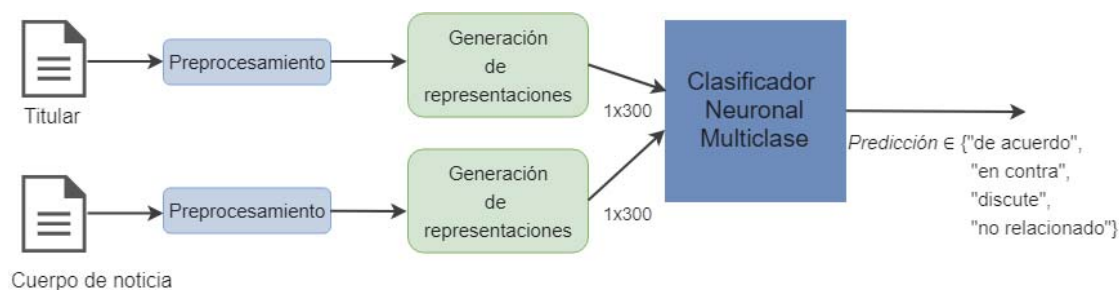


Figura 16: Flujo de procesamiento de muestras.



## 4. Resultados

En esta sección se documenta el proceso de desarrollo y testeo del clasificador de noticias. El desarrollo de la herramienta de filtrado de noticias se compone de tres fases: desarrollo de modelos de representación, implementación del modelo de Deep Learning y desarrollo del servicio web.

La primera fase consiste en el análisis de distintas técnicas de representación de textos. El propósito es determinar qué modelo de representación resulta más efectivo en el dominio del problema. Para ello, se ha realizado un análisis de dos técnicas de representación, una basada en el modelo de bolsa de palabras y otra en los vectores de embedding en word2Vec. Cada representación se ha ejecutado con distintas configuraciones, sobre dos particionados distintos de datos, tomados a partir del corpus original de noticias.

La representación más eficiente y el clasificador más eficaz se emplearán en la plataforma final. Se ha desarrollado una interfaz web para que los usuarios puedan enviar noticias a clasificar. El servicio se encuentra expuesto mediante una API REST.

La implementación de los distintos modelos de clasificación, scripts de tratamiento de texto y ejecución de modelos, junto con el extremo servidor del servicio, se puede consultar en el repositorio público del proyecto <sup>6</sup>. También se encuentran disponibles los ficheros de estadísticas en bruto.

### 4.1. Modelos de representación de textos

Seguidamente se exponen los desarrollos y resultados obtenidos durante la fase de experimentación de modelos de representación de textos. En el apartado de *Metodología* se describe el tratamiento de los datos de entrada, modelos de representación considerados y clasificadores empleados. En la sección de *Resultados* (página 55), se recogen los resultados obtenidos, aportando la tasa de aciertos, valor predictivo positivo y recall, además de las matrices de confusión y curvas ROC para cada clase del problema. Finalmente, se presentan las conclusiones extraídas a raíz de la experimentación, en el apartado de *Discusión* (página 62).

#### 4.1.1. Metodología

A continuación se describe la metodología seguida para la experimentación de modelos de representación de texto. La primera tarea realizada consiste en realizar un preprocesamiento de los datos de entrada, con el fin de eliminar caracteres innecesarios y estandarizar los términos que aparecen. Seguidamente se ha implementado el modelo de representación de bolsa de términos y el modelo de vector de medias de Word2Vec. Para este último modelo se ha empleado un modelo de Word2Vec pre-entrenado y otro entrenado con los datos del problema.

---

<sup>6</sup>[https://github.com/ailopera/tensor\\_project](https://github.com/ailopera/tensor_project)

Para validar los modelos se han elaborado dos clasificadores: un modelo de Random Forest y un modelo de red neuronal hacia delante.

**Tratamiento de los datos de entrada.** El proyecto toma como datos de entrada los datasets ofrecidos por el desafío de FakeNews [67]. Los datasets originales se encuentran divididos en dos ficheros, uno con los cuerpos de noticia únicos, y un segundo con el titular asociado, clasificación y id de cuerpo de noticia único. A su vez, se proporcionan tres particiones: entrenamiento, test y competición (utilizado para medir los resultados de los distintos participantes). Para este trabajo se han utilizado los dos primeros.

El primer tratamiento realizado sobre los datos de entrada ha consistido en aplicar una limpieza sobre los textos (titulares y cuerpos de noticia). Dicho tratamiento consiste en limpiar posibles caracteres o secuencias de textos que son superfluos a la hora de generar las representaciones, como los números y URLs. Otro de los tratamientos necesarios consiste en reducir las posibles inflexiones de un término utilizando su forma base (lematización).

Las transformaciones aplicadas son las siguientes:

- **Eliminación de etiquetas HTML de marcado.** Se ha utilizado el módulo BeautifulSoup [51] de Python para detectar etiquetado de HTML susceptible de estar presente en el cuerpo de la noticia.
- **Eliminación de números y símbolos de puntuación.** Los números se han sustituido por la etiqueta NUM y los símbolos de puntuación por un espacio en blanco. También se han eliminado otros elementos como urls. Se han empleado expresiones regulares para tal fin.
- **Conversión de los caracteres a minúsculas.** Para estandarizar cada ocurrencia de un término determinado el primer paso que se ha aplicado es convertir a minúsculas todos los textos a analizar.
- **Tokenización del texto en palabras.** Se convierte el texto de entrada (titular o cuerpo de noticia) de una cadena continua a una lista de palabras. Esta modificación facilita los tratamientos posteriores.
- **Eliminación de Stopwords.** Se ha realizado un filtrado de estos términos partiendo del dataset de stopwords en inglés de NLTK [42].
- **Lematización.** Haciendo uso del módulo Lemmatizer de la librería NLTK, se han convertido los términos a su forma base. También ha sido necesario realizar un etiquetado POS (Part Of Speech), para guiar al analizador indicándole el tipo de término a estandarizar.

**Modelos de Representación a validar** Una vez que disponemos de los datos de entrada limpiados, el siguiente paso es dar con una representación que refleje fielmente cada instancia o muestra. Estas representaciones suponen el punto de entrada

para el clasificador y determinarán en gran medida la eficacia de éste. El clasificador impone como restricción que los vectores de entrada tengan una dimensión fija, por lo que tendremos que tenerlo en cuenta a la hora de dar con una representación de cada texto de entrada.

Para la representación de los datos de entrada se han desarrollado varias opciones. La primera consiste en partir de un enfoque tradicional, haciendo uso de la conocida representación de bolsa de palabras. La representación de bolsa de palabras realiza un conteo de los términos que aparecen en el texto para generar un vector de  $N$  dimensiones.

Otro de los enfoques valorados consiste en utilizar word2vec para la generación de los modelos de representación. El modelo de word2vec asocia un vector de  $N$  dimensiones a cada término de vocabulario, y nos permite obtener una representación de los textos como una lista de vectores. Ya que tanto los cuerpos de la noticia como los artículos tienen una longitud variable, tenemos que obtener una representación de longitud fija, partiendo de las representaciones de los términos. Para tal fin se ha generado una representación del texto aplicando una media de los vectores de términos del texto.

Para evaluar cuál de los modelos de representación es mejor para utilizarlo en el clasificador final se han desarrollado dos modelos iniciales de clasificación, que nos darán unas medidas iniciales en términos de **tasa de aciertos**, **valor predictivo positivo** y **especificidad**.

**Modelo de bolsa de palabras.** La primera solución abordada es el modelo de bolsa de palabras. Para la construcción del modelo y las representaciones se ha utilizado la clase *CountVectorizer* de Scikit Learn [53], que implementa el modelo de Bolsa de Palabras.

Para la construcción del modelo partimos de corpus de entrada con los preprocesamientos y limpiezas expuestas anteriormente. El programa desarrollado parte del conjunto de entrenamiento, formado por titulares y cuerpos de noticias e inicializa el modelo de Bolsa de palabras de 300 características (términos). Utilizando el método *fit* de *CountVectorizer* [53] se crea el modelo, y mediante el método *transform* se construyen por separado las representaciones de entrenamiento y pruebas para los artículos y cuerpos de noticia.

Para la creación del modelo se ofrecen varios parámetros, enfocados al preprocesamiento y análisis (tokenización, filtrado de stopwords, construcción de ngrams) y al filtrado de términos en base a su ocurrencia en el texto. Debido a que los textos ya han sido tratados antes de ser utilizados descartamos el primer grupo de parámetros, y centramos la experimentación en estudiar distintas configuraciones para el filtrado de términos.

Las opciones de filtrado nos permiten establecer una cota mínima y máxima de ocurrencia. La primera recibe el nombre de *min df*, o *cutoff*, y expresa el porcentaje mínimo de documentos del corpus sobre los que debe aparecer un término para pertenecer al modelo. La segunda medida, *max df*, representa la ocurrencia máxima que puede tener un término a lo largo del corpus de análisis, y nos permite filtrar lo



que se conoce como las stop words específicas del corpus, términos muy frecuentes en datasets de una determinada temática o dominio. Los dos parámetros toman valores flotantes de 0.0 a 1.0, si deseamos expresar la medida en base al porcentaje, o valores enteros si queremos expresarlo de forma absoluta (número de documentos).

Se han empleado cinco configuraciones distintas para los filtrados. La primera consiste en no realizar ningún filtrado, se trata de la configuración por defecto que fija el vectorizador si no se especifica ningún valor. A continuación, se ha experimentado con dos configuraciones en las que se prueba por separado a filtrar los términos con baja ocurrencia o con alta ocurrencia, estableciendo un valor prudencial. Finalmente se han explorado dos configuraciones filtrando en ambos casos, en la primera estableciendo un rango medio de filtrado, en el que se mantienen los términos con una ocurrencia entre el 10 % y el 80 %, y una segunda con unos valores más restrictivos, que únicamente mantienen el 50 % de los términos (en el rango de ocurrencia del 25 % al 75 %).

Se han evaluado cinco configuraciones distintas para la representación, concretamente la frecuencia mínima (min df) y máxima (max df) que debe tener un término para formar parte del modelo. El objetivo de la experimentación era validar si el comportamiento del modelo mejora si se realiza algún tipo de criba para construir la representación, o si por el contrario resulta más provechoso no realizarla.

Los umbrales de frecuencia determinan el tamaño del vector de features. En todas las configuraciones se fija un tamaño máximo de vector de 300 dimensiones, pero no todas son capaces de alcanzar este valor, lo que condiciona el comportamiento del clasificador. La dimensionalidad es un reflejo del vocabulario del que es consciente el modelo a entrenar.

Durante la fase de experimentación se pretenden validar las siguientes hipótesis:

1. Si las representaciones tienen mayor dimensionalidad, se puede particularizar mejor cada texto del dataset. Las mejores configuraciones serán aquellas que consigan obtener el tamaño máximo de representación fijado.
2. La eliminación de términos en base a su alta o baja ocurrencia mejoran el desempeño de la representación. Se desea validar cuál es la configuración más adecuada.

**Modelos Basados en Word2Vec.** Para crear una representación basada en word2vec es necesario partir de un modelo que defina los vectores de embedding asociados a cada texto. Para ello, tenemos dos opciones, utilizar el modelo pre-entrenado proporcionado por los desarrolladores de la plataforma, o entrenar nuestro propio modelo.

La mayor diferencia entre las dos alternativas es el tamaño del modelo y el impacto que tendrán en el desempeño del sistema. El modelo pre-entrenado de Word2vec ha sido construido a partir del dataset de Noticias de Google, que tiene alrededor de 100 billones de términos. El tamaño de vocabulario del modelo es de 3 millones de términos, y tiene una capacidad de 4 GB, un dato para tener en cuenta, ya que

el programa de generación de representaciones tendrá que cargarlo en memoria en tiempo de ejecución.

La carga y acceso a datos del modelo se realiza por medio de la implementación de Word2Vec del paquete gensim [49]. La librería de Word2Vec nos proporciona métodos para cargar el modelo en memoria, y más tarde acceder a él por medio de un array de *numpy* con todos los vectores de embedding.

Otras operaciones interesantes que nos ofrece es el método *doesn't match*, que nos permite realizar preguntas de razonamiento análogo o most similar, que halla los términos más cercanos a uno dado según la distancia de cosenos. Se ha realizado una exploración inicial del modelo en base a dichos términos, como se puede ver en la captura.

<pre> &gt; Global: &gt; globally : 0.6892678737640381 &gt; worldwide : 0.6816036105155945 &gt; theglobal : 0.6362361311912537 &gt; gobal : 0.5978432893753052 &gt; Global : 0.588388979434967 &gt; aglobal : 0.5814876556396484 &gt; world : 0.5744006633758545 &gt; international : 0.541315495967865 &gt; glo_bal : 0.5405515432357788 &gt; gloabl : 0.5397247076034546 </pre>	<pre> &gt; Obama: &gt; mccain : 0.7319011688232422 &gt; hillary : 0.7284600138664246 &gt; obamas : 0.7229631543159485 &gt; george_bush : 0.720567524433136 &gt; barack_obama : 0.7045838832855225 &gt; palin : 0.7043113708496094 &gt; clinton : 0.6934448480606079 &gt; clintons : 0.6816835403442383 &gt; sarah_palin : 0.6815145015716553 &gt; john_mccain : 0.6800708174705505 </pre>
<pre> &gt; War: &gt; wars : 0.748465895652771 &gt; War : 0.6410670280456543 &gt; invasion : 0.5892109870910645 &gt; Persian_Gulf_War : 0.5890660285949707 &gt; Vietnam_War : 0.5886474847793579 &gt; Iraq : 0.588599443435669 &gt; unwinnable_quagmire : 0.5681803226470947 &gt; un_winnable : 0.5606350898742676 &gt; occupation : 0.5506216287612915 &gt; conflict : 0.5506187677383423 </pre>	<pre> &gt; Queen: &gt; queens : 0.7399442791938782 &gt; princess : 0.7070531845092773 &gt; king : 0.6510956883430481 &gt; monarch : 0.6383601427078247 &gt; very_pampered_McElhatton : 0.6357026696205139 &gt; Queen : 0.6163408160209656 &gt; NYC_anglophiles_aflutter : 0.6060680150985718 &gt; Queen_Consort : 0.5923795700073242 &gt; princesses : 0.5908074975013733 &gt; royal : 0.5637185573577881 </pre>
<pre> &gt; Money: &gt; monies : 0.7165061235427856 &gt; funds : 0.7055203318595886 &gt; moneys : 0.6289055347442627 &gt; dollars : 0.628852367401123 &gt; cash : 0.6151220798492432 &gt; vast_sums : 0.6057385802268982 &gt; fund : 0.5789710283279419 &gt; Money : 0.5733489990234375 &gt; taxpayer_dollars : 0.5693671703338623 &gt; Monies : 0.5586516857147217 </pre>	<pre> &gt; Love: &gt; loved : 0.6907792091369629 &gt; adore : 0.6816873550415039 &gt; loves : 0.661863386631012 &gt; passion : 0.6100709438323975 &gt; hate : 0.600395679473877 &gt; loving : 0.5886635780334473 &gt; Ilove : 0.5702950954437256 &gt; affection : 0.5664337873458862 &gt; undying_love : 0.5547305345535278 &gt; absolutely_adore : 0.5536840558052063 </pre>

Figura 17: pruebas sobre el modelo oficial de Word2Vec, utilizando el método most similar.

La segunda alternativa barajada es crear un modelo de word2vec a partir de los datos de entrenamiento (titulares y cuerpos de noticia asociados). El objetivo de utilizar nuestro propio modelo es ver si con los datos de entrada podríamos entrenar un modelo con suficientes términos y que fuera significativo para el tipo de textos que va a analizar el clasificador.

El paso previo a la creación del modelo es la carga y la transformación de los datos de entrada a una matriz, que representa las frases y los términos por frase. Para identificar las distintas frases dentro del texto se ha utilizado el tokenizador basado en símbolos de puntuación (punctk tokenizer) de NLTK [41].

Una vez obtenidas las frases del texto, se ha realizado la limpieza y tratamientos que se expusieron anteriormente. En este caso se ha optado por no quitar los stopwords del texto, ya que el propio algoritmo de word2vec hace un downsampling de los términos comunes, y además hace uso del contexto para obtener el vector de embedding para cada palabra.

El siguiente paso consiste en el entrenamiento del modelo. La parte más importante en este punto es la elección de los hiperparámetros. Los parámetros disponibles se detallan a continuación:

- **Número de dimensiones (size)** Dimensiones de los vectores de representación de cada palabra. Se ha fijado en 300, como está configurado en el modelo preentrenado oficial.
- **Conteo mínimo de palabras (min words)** Ocurrencia mínima que debe tener una palabra para aparecer en el modelo. Valores más bajos dan lugar a modelos con más términos, pero posiblemente desembocan en una representación con más ruido. Se partió de un valor inicial de 40, y tras varias pruebas se fijó en 15, consiguiendo duplicar el tamaño del modelo.
- **Número de trabajadores (workers).** Número de hilos en la ejecución del modelo. Se ha fijado en 4, ya que el servidor donde se entrena el modelo tiene 4 cores.
- **Tamaño de ventana/contexto (window)** Se ha fijado el valor en 10.
- **Downsampling (sample).** Indica el porcentaje de palabras con mayor ocurrencia (stopwords) se descartarán del modelo. Se ha optado por un factor de 0.0001.

La configuración final se ha obtenido realizando una exploración iterativa sobre los modelos generados. El modelo resultante tiene 4259 términos, y cada término se encuentra representado mediante un array de 300 dimensiones. El modelo dispone de un vocabulario reducido, debido al tamaño del dataset de entrada. Si partimos de un modelo propio, tendríamos que considerar un dataset con un mayor número de instancias.

Sobre este modelo se ha realizado una exploración similar al modelo anterior. Observando los términos más cercanos a uno dado encontramos diferencias con respecto al modelo oficial, pero en términos generales se obtienen términos similares. En este caso, al trabajar sobre un dataset más reducido obtenemos datos más particularizados al dominio del problema. Destacamos el caso de “money” y “queen”. En estos casos modelo oficial devuelve unos términos más acertados a nivel semántico, mientras que el modelo propio devuelve los términos que aparecen frecuentemente en conjunción con dichos términos.

La diferencia de este modelo propio con respecto al original es que obtendremos un modelo más sesgado hacia la temática de las noticias que hemos utilizado para el entrenamiento. Esto puede dar lugar a una diferencia mayor en la precisión del

<pre> &gt; Global: &gt; ignite : 0.628195583820343 &gt; march : 0.5895659923553467 &gt; airliner : 0.5821933746337891 &gt; french : 0.5821583271026611 &gt; approximately : 0.5778740644454956 &gt; angela : 0.5774855613708496 &gt; jj : 0.5721689462661743 &gt; abduction : 0.5717968940734863 &gt; inc : 0.5675073862075806 &gt; ad : 0.5567682981491089 </pre>	<pre> &gt; Obama: &gt; wfb : 0.7007851600646973 &gt; administration : 0.6765334606170654 &gt; michelle : 0.6691286563873291 &gt; barack : 0.6625439524650574 &gt; blur : 0.6496133804321289 &gt; president : 0.6419381499290466 &gt; vow : 0.6081100106239319 &gt; gertz : 0.6063179969787598 &gt; king : 0.6048789024353027 &gt; denounces : 0.5874277353286743 </pre>
<pre> &gt; War: &gt; memorial : 0.7428481578826904 &gt; abrams : 0.7132705450057983 &gt; summer : 0.6935235857963562 &gt; j : 0.6913574934005737 &gt; jj : 0.6808469295501709 &gt; awaken : 0.6658874154090881 &gt; nouveau : 0.6512630581855774 &gt; lancement : 0.6464906930923462 &gt; awakens : 0.6453931331634521 &gt; du : 0.6363040804862976 </pre>	<pre> &gt; Queen: &gt; garden : 0.9220095872879028 &gt; elizabeth : 0.9035859107971191 &gt; buckingham : 0.8750764727592468 &gt; mushroom : 0.8519033789634705 &gt; magic : 0.8471107482910156 &gt; palace : 0.8419433832168579 &gt; titchmarsh : 0.8270322680473328 &gt; psychedelic : 0.8205053806304932 &gt; fungi : 0.7387102842330933 &gt; red : 0.7294762134552002 </pre>
<pre> &gt; Money: &gt; beggar : 0.6739267706871033 &gt; food : 0.6480258107185364 &gt; surprise : 0.6412478685379028 &gt; alcohol : 0.5989274978637695 &gt; cash : 0.5894597768783569 &gt; else : 0.5867423415184021 &gt; paint : 0.5817302465438843 &gt; thing : 0.5734326839447021 &gt; gift : 0.5722148418426514 &gt; chicken : 0.5713645219802856 </pre>	<pre> &gt; Love: &gt; halftime : 0.6293606162071228 &gt; desperation : 0.6178413033485413 &gt; drink : 0.5965508818626404 &gt; comfort : 0.5862692594528198 &gt; soup : 0.5860620141029358 &gt; lover : 0.5805528163909912 &gt; girlfriend : 0.5778939723968506 &gt; snake : 0.5650926828384399 &gt; paradise : 0.5568526983261108 &gt; husband : 0.5559821724891663 </pre>

Figura 18: pruebas sobre el modelo propio, utilizando el método most similar

clasificador que se obtenga en la fase de desarrollo con respecto a la obtenida en la fase de producción. Tendremos en cuenta este modelo entre los parámetros de configuración de los modelos de representación para medir su desempeño con respecto al modelo pre-entrenado oficial.

**Vector Average.** La representación basada en vectores de medias parte de los vectores de embedding de los términos del documento y obtiene una representación para el texto a partir de la media de los vectores de términos. Este proceso genera como salida un vector de 300 features para los titulares y otro de 300 para las noticias.

El proceso de obtención de los vectores consiste en cargar el modelo de word2vec a utilizar, formatear los datos de entrenamiento y test, obtener los vectores de características y ejecutar el clasificador determinado. El formateo de textos consiste en transformar la entrada en una lista de listas, que representan la lista de frases del documento.

Una vez transformados los textos al formato deseado, es el momento de obtener los vectores de características para el conjunto de entrenamiento y test. Para obtener el vector de medias, se suman los vectores de embedding del texto de entrada (titular o noticia) y se dividen entre el número total de vectores que representan el

documento.

El último paso consiste en ejecutar el clasificador para probar la calidad de las representaciones. Antes de alimentar el modelo, aplicamos sobre la entrada un transformador para rellenar posibles valores vacíos o incorrectos, generados en el cálculo de medias. Los modelos asumen que todas las entradas son no nulas y de tipo numérico, y para satisfacer esta restricción, tenemos la opción de filtrar las entradas no válidas o inferirlas. Resulta mejor opción decantarnos por la inferencia, con el objetivo de no perder muestras para el análisis. Para la inferencia se utiliza la clase *Imputer* de Scikit-Learn [53], y la estrategia a la hora de generar las representaciones que faltan es aplicando la media de los vectores de características del resto de entradas.

En la práctica, se ha observado que es muy poco frecuente que las representaciones se generen de forma errónea. Sobre el conjunto de datos de entrada únicamente se ha generado una representación incorrectamente, pero se ha optado por mantener el uso del modelo de inferencia de datos.

El programa desarrollado para tal fin permite ejecución por línea de comandos o mediante otros scripts, y las opciones configurables son el modelo de word2vec, el clasificador (perceptrón multicapa o random forest) y las particiones de datos de entrada. Si no se especifican datasets de entrada, el programa toma uno por defecto para realizar una ejecución de prueba.

Para optimizar el proceso de formateo de textos, se ha utilizado la librería JobLib [65], que partiendo del módulo de multiprocesamiento de Python, paraleliza esta tarea entre los núcleos del servidor. Se ha tratado de paralelizar la construcción de vectores de medias, pero finalmente se ha descartado, ya que resultaba muy poco óptima cuando se utilizaban modelos de gran tamaño, como el oficial. Cada núcleo requiere cargar en memoria el modelo completo, por lo que se genera un cuello de botella que realentiza considerablemente el tiempo de ejecución. La versión secuencial de esta fase en concreto obtenía unos tiempos de cómputo significativamente menores y requiere menor memoria.

La mayor desventaja de los vectores de medias es que se pierde la información del contexto, como ocurre con el modelo de bolsa de palabras. Se parte de lo que se supone como una representación más fiel de los términos, pero al realizar la media se difumina la caracterización de cada término. Al considerar todos los términos para realizar la media se pierde la noción del contexto.

Con la experimentación sobre el modelo de vectores de medias se pretende validar lo siguiente:

1. Las representaciones basadas en word2vec suponen una mejora con respecto al modelo clásico de Bolsa de términos, en términos de tasa de aciertos, valor predictivo positivo y *recall*. El clasificador entrenado con vectores de medias tendrá una capacidad de generalización mejor.
2. Analizar si resulta más provechoso utilizar un modelo preentrenado, con una representación más completa y general del vocabulario, o uno específico entrenado con textos del dominio del clasificador, menos términos y con una

representación más particular de estos.

3. Analizar si aplicar la media de representaciones embebidas perjudica la calidad de la representación resultante.

**Modelos de Machine Learning utilizados en la validación.** Los modelos se han validado utilizando un clasificador basado en métodos clásicos y otro basado en redes de neuronas artificiales. El primer modelo consiste en un RandomForest, implementado con la librería Scikit-Learn [53] y el segundo en un clasificador multi-capas implementado con TensorFlow [61]. El objetivo de este estudio es el de obtener unos resultados base que nos servirán para determinar qué representación resultaría más efectiva en el clasificador final.

Estos modelos tomarán exclusivamente como entrada los modelos de representación, ya que en este punto de la experimentación queremos evaluar las representaciones únicamente. Lo que nos interesa es averiguar cuál de las representaciones consigue un score mejor, en dos clasificadores base definidos.

En ambos modelos ha sido necesario transformar las clases objetivo tanto de entrenamiento como de test a variables enteras categóricas, con la siguiente correspondencia:

- **Agree:** 0.
- **Disagree:** 1.
- **Discuss:** 2.
- **Unrelated:** 3.

**Modelo de Random Forest.** El modelo de Random Forest [7] forma parte de los conocidos métodos de aprendizaje conjunto (Ensemble Methods) [?], que se basan en la utilización de varios estimadores para predecir las clases de salida. Un bosque aleatorio se encuentra formado por un conjunto de árboles de decisión, y la clase de salida es el resultado de realizar una media de las clasificaciones obtenidas por cada estimador.

El clasificador de Random Forest implementado se ha configurado con **100 estimadores** y se han empleado los métodos de *fit* y predicción, disponibles en el paquete *RandomForestClassifier* de **Scikit Learn** [53]. Dicho clasificador toma como entrada los vectores de representación de titulares y noticias, como una lista de vectores de 1x600 features.

Para el cálculo de métricas se utiliza la librería metrics de Scikit-Learn [53]. Para el cálculo de las métricas de precisión y recall, se ha utilizado el valor “weighted” como configuración del cálculo de la media. Esta opción aplica una media ponderada del valor obtenido en cada clase para cada métrica. De esta forma se tiene en cuenta el sesgo que presenta el dataset en cuanto a instancias de clasificación. Antes de probar con esta configuración se probó la versión “micro” del cálculo, pero se terminó descartando porque daba lugar a unos valores de precision y recall idénticos con



respecto a la tasa de aciertos. La opción *micro* calcula las métricas de forma global, independientemente de la clase.

**Modelo de Perceptrón Multicapa.** El segundo modelo utilizado para el testeo consiste en una red neuronal hacia delante con dos capas ocultas, desarrollado con la librería **TensorFlow** [61]. Las redes de neuronas ofrecen una gran flexibilidad de configuración de los modelos, ya que es necesario tomar múltiples decisiones de diseño para su creación. A su vez esta característica hace que resulte complejo obtener unos resultados óptimos sin estudiar cuál es la configuración adecuada para los hiperparámetros del modelo. Esta arquitectura pretende ser un ejemplo simplificado de como se comportarían las representaciones en un modelo de Deep Learning. En una siguiente fase del trabajo se optimizará este modelo (sección 4.2), por lo que los hiperparámetros que se exponen a continuación se corresponden con la **configuración inicial** del modelo.

La capa de entrada está formada por **600 neuronas**, para recibir los vectores de features que representan los titulares y noticias respectivamente. Se ha optado por configurar un número decreciente de neuronas a lo largo de las capas ocultas. Las neuronas de una capa oculta se encuentran completamente conectadas con las neuronas de la siguiente. La primera capa oculta se compone de 300 neuronas, y la segunda de 100, y en las dos se emplea la **función de ReLu** como activación.

La capa de salida se encuentra formada por 4 neuronas, y cada una representa la probabilidad que tiene una determinada instancia de pertenecer a una clase concreta. Sobre las probabilidades se aplica una **función de Logit** y una función de normalización. En problemas de clasificación se suele emplear como función de normalización la función de softmax, concretamente, la versión de **entropía cruzada** en clasificaciones no binarias.

Para el entrenamiento se emplea la función del **Gradiente Descendiente**, con una tasa de aprendizaje de 0.01. Se ha fijado un número de epochs e iteraciones constantes, en base al tamaño de los datos de entrada. El entrenamiento está compuesto por 20 epochs, y en cada uno se va entrenando iterativamente el modelo con un subconjunto de los datos de entrada.

Otro de los parámetros a la hora de definir el entrenamiento qué porcentaje de muestras se toman en cuenta en cada iteración. Para ello existen tres versiones del algoritmo del Gradiente: la versión clásica, la versión Mini-Batch y la versión estocástica. Para este modelo se ha optado por la versión de **Mini Batch**, en cada iteración ajusta la red en base a conjuntos de 50 muestras, tomadas aleatoriamente a partir del conjunto de entrenamiento. Esta variante permite entrenar modelos en menor tiempo y consume menos recursos computacionales.

Al final de cada epoch, se computa la tasa de aciertos sobre los datos de entrenamiento y de test, y al final del entrenamiento se obtiene el valor sobre el dataset completo de entrenamiento y test.

El modelo da como salida un vector de *logits*, que representa la probabilidad de pertenencia a las cuatro clases para la muestra de entrada proporcionada (modelo de regresión logística). La predicción final del clasificador es aquella que presenta

una mayor probabilidad. En TensorFlow se obtiene mediante los métodos *in\_top-k* y *argmax*[61].

Después la ejecución del modelo se obtienen las métricas de recall, valor predictivo positivo, matriz de confusión y curvas ROC. En la medición la mayor complicación consiste en asegurar que tanto el modelo de Random Forest como este modelo están generando unas métricas equiparables. Para este modelo se emplearon inicialmente los métodos de cálculo ofrecidos por la API de TensorFlow [61], pero tras algunas pruebas, se comprobó que dicha implementación no tenía en cuenta el desnivel de instancias de cada clase. Finalmente se ha utilizado el mismo método y parámetros de la librería Scikit Learn, y se ha observado una diferencia en que varía del 10 % al 30 % en el valor de los resultados con respecto la versión de TensorFlow.

Las funciones de la API de Tensorflow [61] han facilitado la creación de este modelo, como *tf.layers.dense*, que define el esquema de interconexión entre capas, y las funciones para definir la función de optimización (*tf.train.GradientDescentOptimizer*) y entropía (*tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits*).

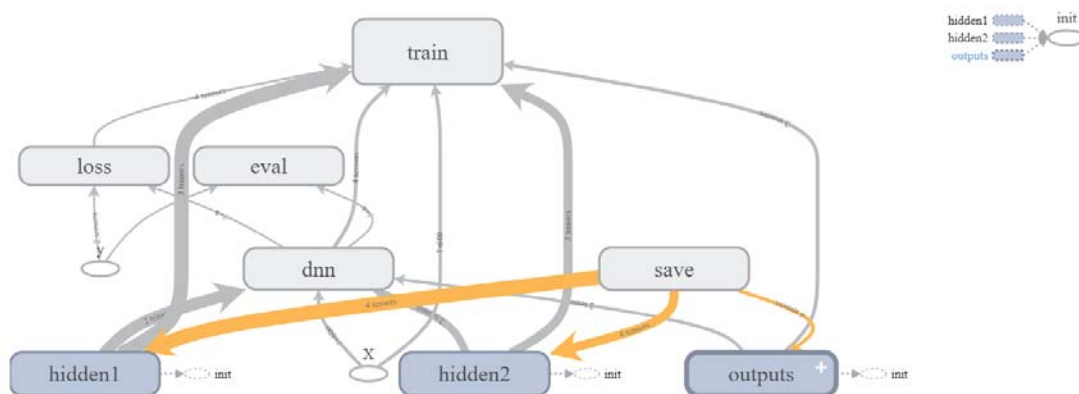


Figura 19: Grafo de computación del modelo de Perceptrón Multicapa

Particionado de datos de entrada. Métricas de evaluación del modelo.

Para el testeo de las representaciones se ha seguido el procedimiento de validación simple. Para obtener las particiones de entrenamiento y validación se ha realizado una división del 60 %-20 % del dataset completo. Estos datos se han empleado para ejecutar todas las configuraciones que se pretendían validar. Las mejores configuraciones se han contrastado con la partición de test, que suponen el 20 % restante.

Se han tomado las particiones originales de entrenamiento y test ofrecidas por el desafío [67], y se han fusionado en un fichero para realizar dos particionados distintos. Sse ha implementado un script que toma los datasets originales, aplican las transformaciones sobre el texto establecidas y generan el fichero con los datos fusionados. A partir de este fichero intermedio se obtienen los ficheros de entrenamiento/validación y test para los dos particionados definidos.



Una muestra está compuesta por los siguientes campos:

- Titular preprocesado de la noticia.
- Cuerpo de la noticia preprocesado.
- Clasificación de la noticia: *Agree*, *Disagree*, *Discuss* o *Unrelated*.
- Id del cuerpo de noticia. Incluido como dato de control, no utilizado en la clasificación.

La métrica inicialmente considerada para la validación fue la de accuracy sobre el conjunto de datos de entrenamiento y de test. Sin embargo, en problemas de clasificación la medida de accuracy no nos da suficiente información sobre la efectividad del modelo, y es necesario contrastarla con otras métricas que nos permiten analizar el desempeño en otros términos. La precisión de una ejecución en particular no tiene valor por sí sola, ya que el conjunto de datos de entrenamiento o test pueden estar sesgados hacia una determinada clase, y el modelo de salida puede ser capa de detectar con un ratio muy alto dicha clase, y a su vez detectar con menor acierto el resto de clases.

Para obtener una visión más completa sobre el clasificador se ha medido la especificidad y sensibilidad. La primera se corresponde con la métrica de precisión, que mide la proporción de positivos correctos con respecto a la suma de positivos totales (accuracy de las predicciones correctas). La sensibilidad viene dada por la métrica de recall, que obtiene el ratio entre positivos correctos con respecto a positivos correctos y falsos negativos.

Para las configuraciones finales se exponen las **matrices de confusión** y **curvas ROC**, que aportan una perspectiva gráfica de las métricas obtenidas. Las columnas de la matriz de dispersión expresan las clases reales, y las filas, las clases predichas por el clasificador. La diagonal principal representa los aciertos (verdaderos positivos) del clasificador. Si el clasificador detectara correctamente las clases de cada muestra analizada generaría una matriz con valores sólo en la diagonal. La matriz de confusión ofrece una manera gráfica de observar el desempeño del clasificador para cada clase, y nos aporta una perspectiva más completa, más allá de los valores de precision y recall obtenidos.

Las curvas ROC representan el ratio de verdaderos positivos frente al ratio de falsos positivos para varios niveles de discriminación, y es independiente de la distribución de clases, por lo que constituye una medida eficaz para evaluar la eficiencia del modelo. Para generar la curva ROC es necesario obtener las probabilidades de pertenencia a una clase que genera el clasificador al predecir una muestra (*Y\_score*). En el modelo de Random Forest esto se obtiene a partir de la función `predict_proba` de la API de Scikit-Learn [53], mientras que en el modelo de red neuronal es el vector de *logits* que da como salida el clasificador.

El script de validación implementado se encuentra parametrizado en base al dataset de entrenamiento/validación + test a utilizar, y al modelo que se desee validar. En base al modelo a validar se ha aplicado un conjunto de configuraciones

determinado, específicas del modelo. Dicho script también se encarga de realizar la división de datos de entrenamiento/validación.

**Particionado 1.** El primer particionado realiza una partición simple del 60 %-20 %-20 % de los datos de entrada, aplicando una aleatorización previa de las muestras.

Cada muestra de clasificación existe en un único dataset. No obstante, en este particionado no se ha tenido en cuenta que un cuerpo de noticia aparezca en una sola partición. Para el entrenamiento se utiliza un total de 994 cuerpos de noticias distintos.

El número de instancias en cada conjunto es el siguiente:

- **Número de muestras de entrenamiento:** 39220.
- **Número de muestras de validación:** 9805.
- **Número de muestras de test:** 10935.

Se ha observado que los datos de entrada se encuentran altamente desbalanceados en favor de una clase. Existe una proporción muy alta de ejemplos de noticias no relacionadas y a su vez una proporción muy baja para los ejemplos de noticias no relacionadas, un dato a tener en cuenta a la hora de desarrollar el modelo. Este fenómeno complica la obtención de un modelo con una caracterización equiparable de todas las clases.

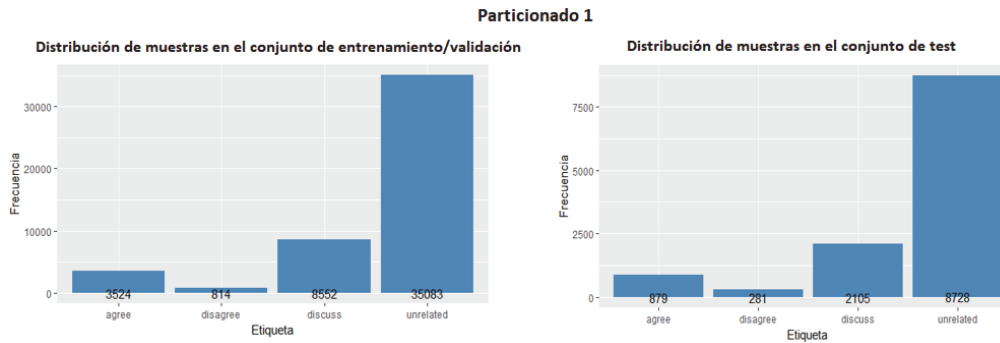


Figura 20: Clasificaciones de las particiones de entrenamiento y test (Particionado 1)

**Particionado 2.** Se ha definido un segundo particionado en el que se asegura que los cuerpos de noticia se encuentran en el conjunto de entrenamiento/validación o en el de test.

Para lidiar con el problema del desnivel de muestras de clasificación, en tiempo de ejecución se aplica un método de oversampling. En concreto se ha empleado el método **SMOTE** (Synthetic Minority Over-sampling Technique)[8], que crea muestras sintéticas de clases poco representadas en base a las muestras existentes. Se

ha utilizado la implementación de la librería de Python *Imbalanced learn* [32]. Los parámetros más importantes a configurar son el algoritmo utilizado por SMOTE para realizar el oversampling y el ratio. Se ha experimentado con el algoritmo regular y el basado en SVM, y finalmente se ha utilizado el primero para la validación de los modelos, ya que el algoritmo de SVM aumentaba considerablemente el tiempo de ejecución. En cuanto al ratio, se han probado dos tipos: sobremuestreo de la clase minoritaria (configuración *minority*) y oversampling de todas las clases (configuración *all*). Durante las pruebas los resultados de sobremuestreo de todas las clases ofrecían unos resultados ligeramente mejores, por lo que se ha utilizado esta configuración para la obtención de resultados. Esta configuración provoca que todas las clases presenten el mismo número de muestras que la clase mayoritaria.

Para la experimentación se ha utilizado la versión sin aplicar *oversampling* y aplicando *oversampling*. El número de instancias original en cada conjunto es el siguiente:

- Número de muestras de entrenamiento: 38379.
- Número de muestras de validación: 9595.
- Número de muestras de test: 11993.

Del conjunto de entrenamiento/validación, se toma un 80 % para entrenamiento y 20 % para validación. Tras aplicar el sobremuestreo sobre el conjunto de test todas las muestras tienen la misma frecuencia, obteniendo un total de 28827 muestras por clase, se consigue la misma frecuencia de la clase mayoritaria. Las muestras artificiales se generan una vez se han obtenido los vectores de representación (bolsa de términos o vector de medias de Word2Vec), antes de ejecutar el clasificador. Sobre los conjuntos de validación y test no se aplica SMOTE. La proporción de clases previa a la aplicación de SMOTE se expone a continuación.

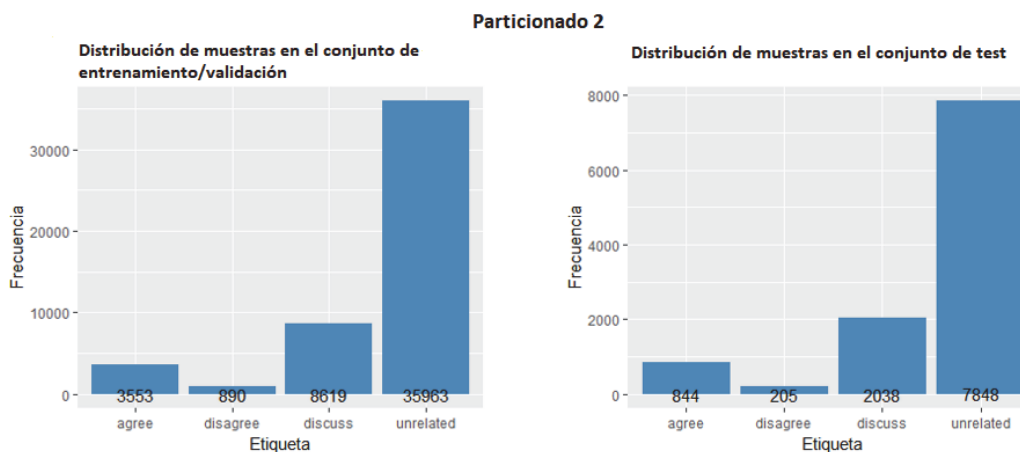


Figura 21: Clasificaciones de las particiones de entrenamiento y test (Particionado 2)

#### 4.1.2. Resultados

A continuación se presentan los resultados cuantitativos de la validación y testeo de modelos de representación, empleando las dos particiones mencionadas.

**Representación Bag Of Words.** Para la representación de Bolsa de términos se presentan los resultados de las 5 configuraciones planteadas, validadas con el modelo de Random Forest y Perceptrón Multicapa.

**Particionado 1.** El rendimiento del clasificador se sitúa entre el 63 % y 91 %, para el valor predictivo positivo, y entre el 74 % y el 92 % en términos de especificidad. La configuración que reportan mejores resultados es la tercera, en la que se aplica un filtrado de términos con una ocurrencia mayor al 80 %. Ambas configuraciones generan vectores de características de 300 dimensiones para el titular y el cuerpo de la noticia. El clasificador basado en Random Forest reporta unos resultados ligeramente superiores sobre la mejor configuración, con respecto a la ejecución en el modelo de perceptrón multicapa.

labelBOWP1Entrenamiento										
Características	T. ejecución	min df	max df	Clasificador	Tasa aciertos (E)	T aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)
600	136.8 s	1	1.0	MLP	0.91	0.89	0.9	0.88	0.91	0.89
140	69.09 s	0.1	1.0	MLP	0.84	0.83	0.83	0.81	0.84	0.83
<b>600</b>	<b>134.74 s</b>	<b>1</b>	<b>0.8</b>	<b>MLP</b>	<b>0.91</b>	<b>0.89</b>	<b>0.9</b>	<b>0.88</b>	<b>0.91</b>	<b>0.89</b>
140	72.34 s	0.1	0.8	MLP	0.83	0.83	0.82	0.81	0.83	0.83
8	53.96 s	0.25	0.75	MLP	0.73	0.74	0.63	0.63	0.73	0.74
600	56.09 s	1	1.0	RF	0.99	0.91	0.99	0.91	0.99	0.91
140	29.79 s	0.1	1.0	RF	0.95	0.86	0.95	0.86	0.95	0.86
<b>600</b>	<b>57.85 s</b>	<b>1</b>	<b>0.8</b>	<b>RF</b>	<b>0.99</b>	<b>0.91</b>	<b>0.99</b>	<b>0.91</b>	<b>0.99</b>	<b>0.91</b>
140	31.72 s	0.1	0.8	RF	0.95	0.86	0.95	0.86	0.95	0.86
8	21.08 s	0.25	0.75	RF	0.81	0.78	0.8	0.76	0.81	0.78

Cuadro 1: Resultados de validación con el modelo de bolsa de palabras, particionado 1

Los dos clasificadores un buen rendimiento, siendo el modelo de MLP el que ofrece un mejor comportamiento en términos de no overfitting, manteniendo unas métricas competitivas. La mejor configuración reportada se ejecuta sobre la partición de test, para obtener el rendimiento final. Se observa que la efectividad del clasificador disminuye, obteniendo unos valores de valor predictivo positivo y especificidad por debajo del 70 %. También se observa overfitting del modelo, por la amplia diferencia entre la tasa de aciertos de los datos de entrenamiento con respecto a los de test.

Características	min df	max df	Clasificador	T. ejecución (s)	T aciertos (T)	Precisión (T)	Recall (T)
600	0.1	0.8	MLP	154.97 s.	0.71	0.64	0.71

Cuadro 2: Resultados de test con el modelo de bolsa de palabras, particionado 1.

Finalmente se obtienen las matrices de confusión y curvas ROC para la configuración final, sobre el conjunto de test, para observar el balance entre el valor predictivo positivo y la especificidad. Los dos modelos presentan problemas para identificar muestras de la clase minoritaria.

Representación de bolsa de palabras (frecuencia máxima 0.8)

Particionado 1

**Modelo de Perceptrón Multicapa**

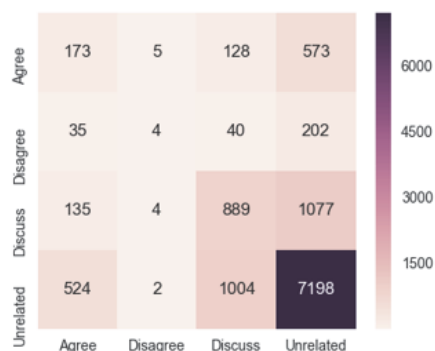


Figura 22: Matrices de confusión de la configuración final para la representación de Bolsa de palabras, particionado 1

El clasificador caracteriza mejor las noticias de acuerdo y no relacionadas con el titular, de acuerdo a su valor de AUC (área bajo la curva). La clase minoritaria es la que se caracteriza de una forma más irregular, con un valor de AUC del 0.65.

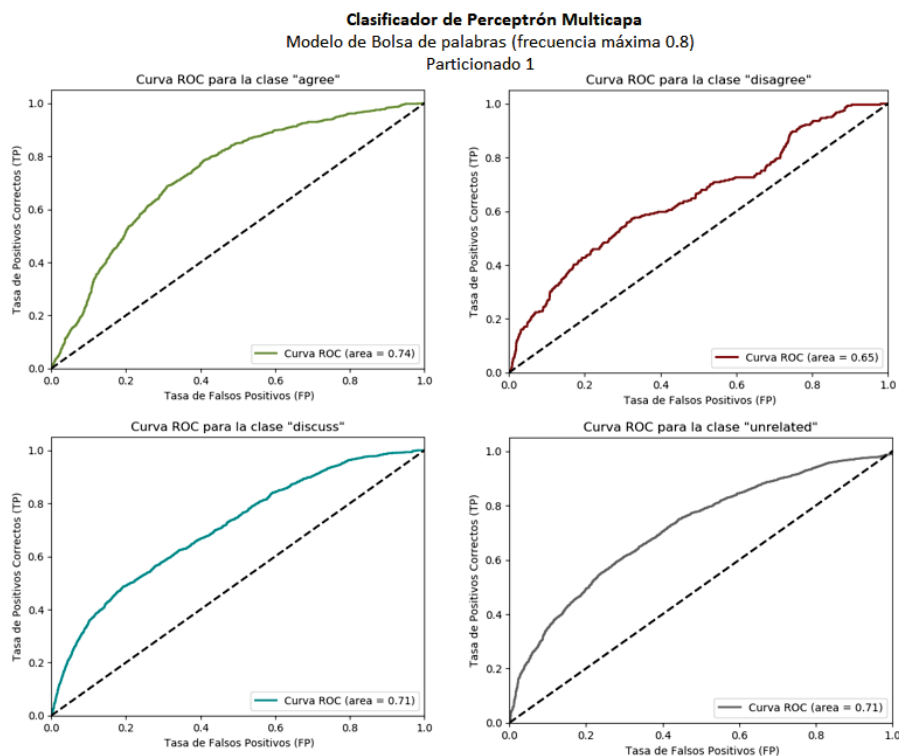


Figura 23: Curvas ROC por clase para el modelo de bolsa de palabras, sobre la partición 1.

**Particionado 2** Sobre esta partición se reportan unos resultados peores y más próximos entre sí. Se validan como mejores configuraciones la primera y la tercera, en ambos clasificadores. La aplicación de oversampling sobre las clases mejora levemente la precisión y sensibilidad, aunque se continúa observando overfitting.

Características	min df	max df	Clasificador	T. ejecución	Tasa aciertos (E)	T aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	SMOTE
600	127.09	1	1.0	MLP	0.91	0.7	0.9	0.67	0.91	0.7	no
138	71.79	0.1	1.0	MLP	0.84	0.69	0.83	0.64	0.84	0.69	no
600	138.49	1	0.8	MLP	0.9	0.72	0.9	0.7	0.9	0.72	no
138	74.53	0.1	0.8	MLP	0.85	0.66	0.84	0.65	0.85	0.66	no
6	55.57	0.25	0.75	MLP	0.74	0.72	0.63	0.58	0.74	0.72	no
600	66.15	1	1.0	RF	0.99	0.76	0.99	0.73	0.99	0.76	no
138	32.24	0.1	1.0	RF	0.95	0.74	0.94	0.7	0.95	0.74	no
600	68.07	1	0.8	RF	0.99	0.77	0.99	0.74	0.99	0.77	no
138	32.62	0.1	0.8	RF	0.95	0.73	0.94	0.69	0.95	0.73	no
6	20.15	0.25	0.75	RF	0.8	0.65	0.79	0.59	0.8	0.65	no
600	384.8	1	1.0	MLP	0.96	0.6	0.96	0.71	0.96	0.6	sí
138	187.67	0.1	1.0	MLP	0.88	0.55	0.88	0.66	0.88	0.55	sí
600	347.07	1	0.8	MLP	0.96	0.63	0.96	0.71	0.96	0.63	sí
138	189.53	0.1	0.8	MLP	0.89	0.54	0.89	0.67	0.89	0.54	sí
6	128.52	0.25	0.75	MLP	0.46	0.33	0.45	0.61	0.46	0.33	sí
600	162.85	1	1.0	RF	1.0	0.71	1.0	0.72	1.0	0.71	sí
138	70.28	0.1	1.0	RF	0.97	0.6	0.97	0.68	0.97	0.6	sí
600	163.81	1	0.8	RF	1.0	0.74	1.0	0.74	1.0	0.74	sí
138	69.89	0.1	0.8	RF	0.97	0.57	0.97	0.67	0.97	0.57	sí
6	29.97	0.25	0.75	RF	0.7	0.4	0.71	0.6	0.7	0.4	sí

Cuadro 3: Resultados de validación con el modelo de bolsa de palabras, utilizando el particionado 2, con SMOTE y sin él.

Sobre la partición de test se mantienen los resultados observados. Se sigue obteniendo overfitting, y la mejor configuración es la tercera (filtrado de términos con una frecuencia normalizada superior al 80 %), aplicando oversampling sobre las clases minoritarias.

Características	min df	max df	Clasificador	T. ejecución (s)	T aciertos (T)	Precisión (T)	Recall (T)
600	1	0.8	MLP	356.98	0.63	0.74	0.63

Cuadro 4: Resultados de test con el modelo de bolsa de palabras, utilizando el particionado 2, con SMOTE.

Con la matriz de dispersión podemos ver en detalle las clasificaciones realizadas sobre el modelo. Disminuye el número de clasificaciones erróneas de textos como noticias no relacionadas. Aumenta la capacidad de detección de noticias en contra del titular.

En las curvas ROC se puede ver una mejora en la detección de las noticias a favor y en contra, dando lugar a una ligera mejora en la detección de noticias no relacionadas (al disminuir el ratio de falsos positivos). No obstante, empeora ligeramente la detección de noticias que discuten al popular, de acuerdo al valor de AUC reportado.

**Modelos basados en Word2Vec. Representación de Vector Average.** Se presentan los resultados de las dos configuraciones consideradas, en base al modelo de Word2Vec empleado, en los dos escenarios de uso planteados (clasificador Random Forest y red neuronal).

Representación de bolsa de palabras (frecuencia máxima 0.8)

Particionado 2

**Modelo de Perceptrón Multicapa**

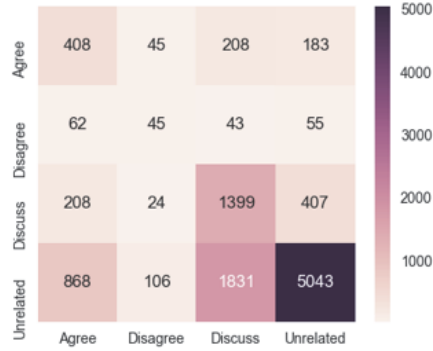


Figura 24: Matrices de confusión de la configuración por defecto para la representación de Bolsa de palabras, sobre la partición 2.

**Clasificador de Perceptrón Multicapa**  
Modelo de Bolsa de palabras (frecuencia máxima = 0.8)  
Particionado 2

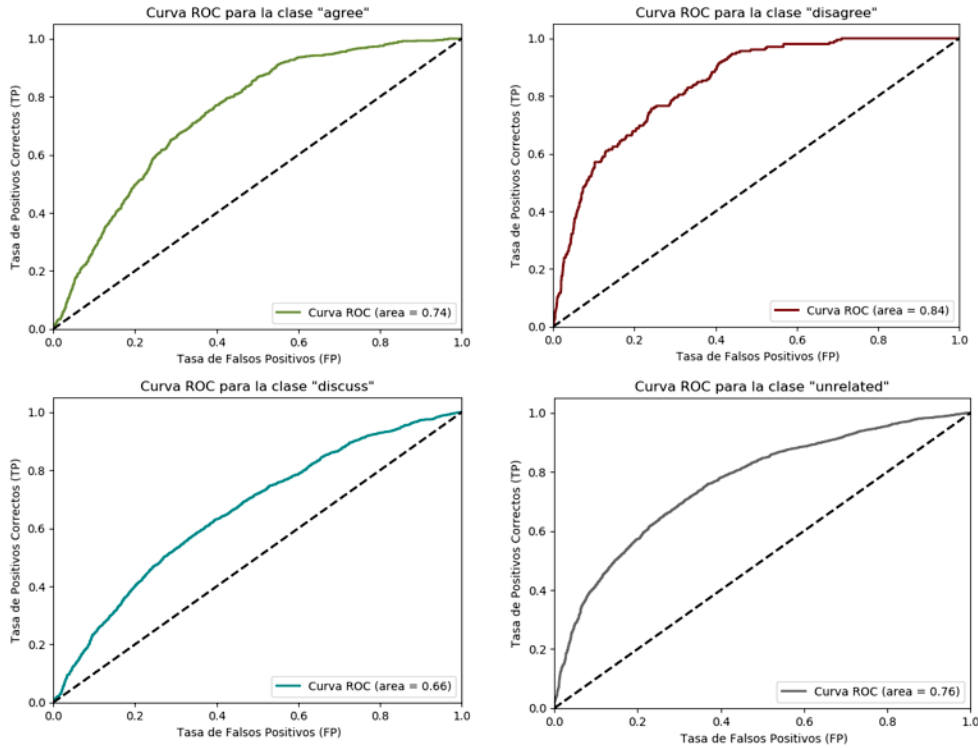


Figura 25: Curvas ROC por clase para el modelo de bolsa de palabras, sobre la partición 2.

**Partición 1.** Para testear el modelo se utiliza el modelo de word2vec entrenado con los textos de entrenamiento, y el modelo pre-entrenado oficial. La ejecución de estos modelos toma un mayor tiempo debido al coste de construir los vectores

de características a partir del cómputo de la media. Esto da lugar que aumente aproximadamente el doble el tiempo de ejecución, situándose de media entorno a los tres minutos, con respecto a la representación de Bolsa de términos.

En este caso se ha obtenido una tasa de aciertos en el rango del 83 %, para el modelo de perceptrón multicapa, y 94 %, para el modelo de Random Forest.

La representación basada en vector de medias ofrece unos resultados mejores en términos generales con respecto al modelo de bolsa de palabras, en la fase de validación. Se consigue un valor predictivo positivo mínimo del 77 % y un *recall* mínimo de 80 %, unas décimas mejor que en el otro modelo de representación. En este caso también se obtienen unos resultados de valor predictivo positivo y especificidad muy a la par dentro del mismo clasificador, independientemente del modelo de word2vec empleado.

En el escenario de uso del modelo pre-entrenado se consigue una precisión algo mayor, aunque con un *overfitting* más pronunciado. Se ejecuta dicha configuración sobre el conjunto de test para obtener las métricas de desempeño final. El clasificador basado en Random Forest reporta unos valores de valor predictivo positivo y especificidad ligeramente por encima del modelo MLP, pero se observa una tendencia mayor al *overfitting*.

Modelo Word2Vec	Clasificador	T. ejecución	Tasa aciertos (E)	T aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)
Propio	MLP	192.6	0.82	0.83	0.78	0.79	0.82	0.83
Propio	RF	152.18	1.0	0.98	1.0	0.97	1.0	0.98
Pre-entrenado	MLP	225.56	0.83	0.83	0.8	0.8	0.83	0.83
Pre-entrenado	RF	187.43	1.0	0.94	1.0	0.94	1.0	0.94

Cuadro 5: Validación del modelo de vectores de medias, partición 1.

Se ejecuta la configuración de modelo de uso pre-entrenado sobre los dos clasificadores con el objetivo de reportar la efectividad final en cada escenario.

Modelo Word2Vec	Clasificador	T. ejecución (s)	T aciertos (T)	Precisión (T)	Recall (T)
Pre-entrenado	MLP	229.24	0.76	0.72	0.76

Cuadro 6: Testeo del modelo de vectores de medias, primera partición

De acuerdo a las matrices de confusión para el conjunto de test, se registra de forma cuantitativa las clasificaciones realizadas. En el modelo de perceptrón multicapa el número de muestras de entrada de la clase *disagree* no es suficiente para que el modelo la considere como tal, por lo que la mayoría de estas noticias se terminan clasificando como no relacionadas. Sin embargo, el modelo de Random Forest si la reconoce como clase, aunque no tiene una gran relevancia en términos de clasificación.

Finalmente, se reportan las curvas ROC de cada clase, para el clasificador neuronal multicapa. Se observan unos resultados de partida superiores con respecto a la representación de bolsa de términos. La clase minoritaria sigue presentando un comportamiento irregular.



**Modelo de Perceptrón Multicapa**  
Modelo Propio (4259 términos, vectores embedding de 300 features)

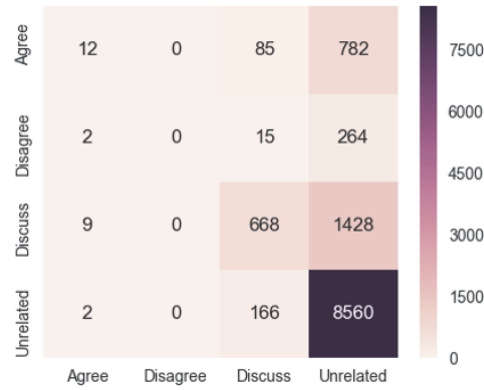


Figura 26: Matrices de confusión sobre el conjunto de test en los modelos de Vectores de Medias

**Clasificador de Perceptrón Multicapa**  
Modelo word2vec pre-entrenado oficial  
Particionado 1

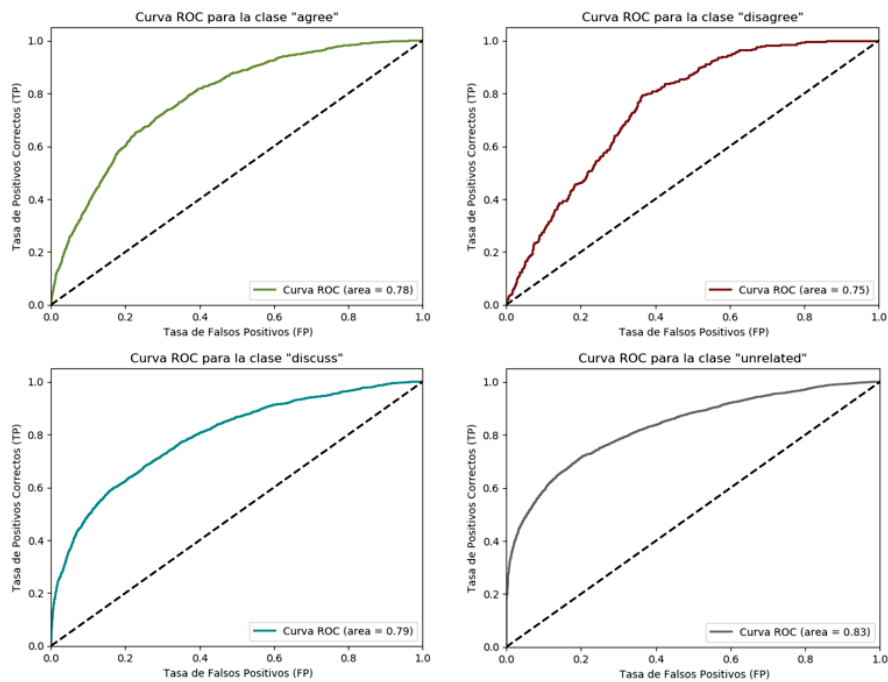


Figura 27: Curvas ROC del modelo de vector de medias, particionado 1

**Particionado 2.** Sobre este particionado se obtienen unos resultados en la franja 75 % y el 80 %, para el valor predictivo positivo y especificidad sobre los datos de validación. Aplicar SMOTE sobre los datos de entrenamiento mejora la sensibilidad y especificidad, en todas las configuraciones. El uso del modelo de word2vec propio ofrece unos resultados similares con respecto al modelo pre-entrenado, aten-

diendo a las configuraciones que aplican SMOTE. El modelo propio presenta una menor diferencia entre las métricas de entrenamiento y validación.

Modelo Word2Vec	Clasificador	T. ejecución	Tasa aciertos (E)	Tasa aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	SMOTE
Propio	MLP	189.88	0.85	0.77	0.81	0.72	0.85	0.77	no
Propio	RF	143.17	1.0	0.8	1.0	0.82	1.0	0.8	no
Pre-entrenado	MLP	211.86	0.82	0.76	0.79	0.7	0.82	0.76	no
Pre-entrenado	RF	178.52	1.0	0.77	1.0	0.8	1.0	0.77	no
Propio	MLP	412.8	0.9	0.8	0.9	0.8	0.9	0.8	sí
Propio	RF	408.79	1.0	0.8	1.0	0.8	1.0	0.8	sí
Pre-entrenado	MLP	421.02	0.97	0.79	0.97	0.79	0.97	0.79	sí
Pre-entrenado	RF	433.22	1.0	0.79	1.0	0.8	1.0	0.79	sí

Cuadro 7: Validación del modelo de vectores de medias, particionado 2

La configuración escogida utiliza el modelo pre-entrenado para construir los vectores, y se aplica SMOTE sobre los vectores de entrenamiento. Los resultados sobre la partición de test reportan un comportamiento equilibrado en términos de sensibilidad y especificidad, obteniendo una tasa de 85 %.

Modelo Word2Vec	Clasificador	T. ejecución (s)	Tasa aciertos (T)	Precisión (T)	Recall (T)	SMOTE
Pre-entrenado	MLP	472.5	0.85	0.86	0.85	sí

Cuadro 8: Testeo del modelo de vectores de medias, particionado 2

Observando las matrices de confusión vemos que gracias al oversampling de las clases minoritarias el modelo es capaz ahora de detectar noticias en desacuerdo, disminuyendo el número de falsos positivos de la clase *unrelated*.

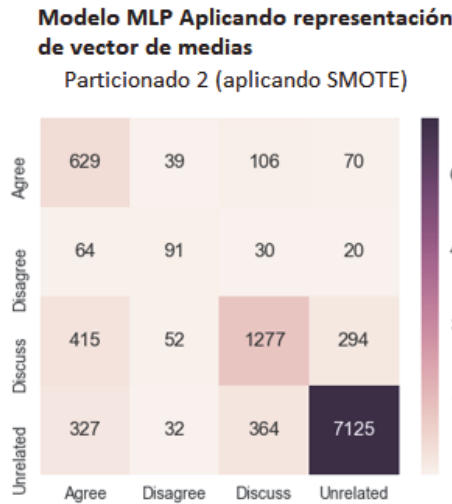


Figura 28: Matrices de confusión sobre el conjunto de test en los modelos de Vectores de Medias, sobre la partición 2

Esta mejora en el clasificador también se hace patente en las curvas ROC, en el que se puede percibir un aumento del área bajo la curva en todas las clases, gracias a la aplicación de SMOTE. Las curvas reportan el mejor resultado hasta el momento,

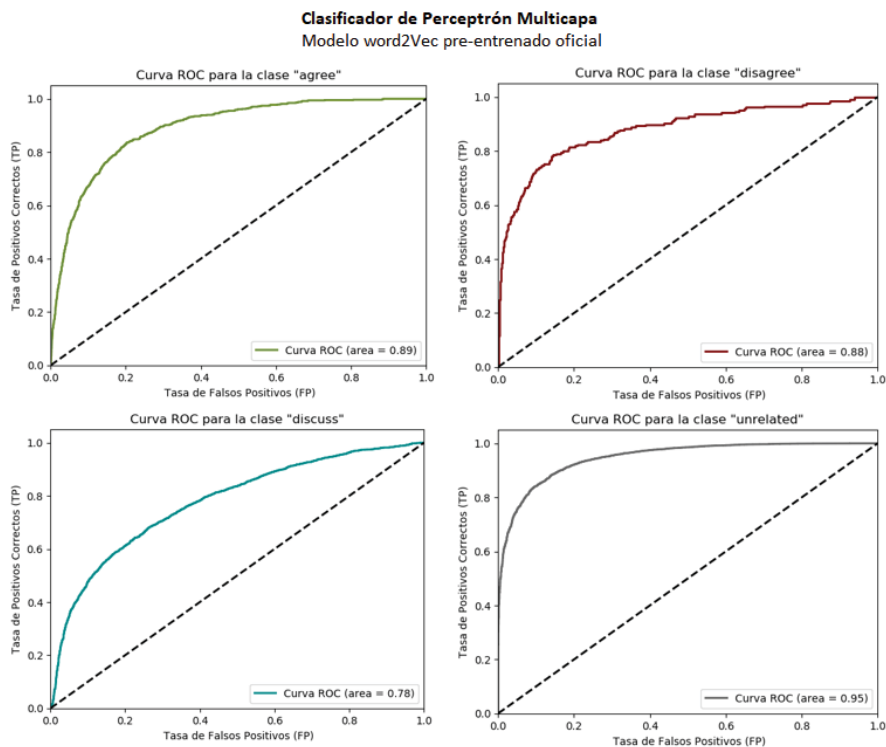


Figura 29: Curvas ROC del modelo de vector de medias, particionado 2

en términos de precisión y *recall*. El comportamiento para cada una de las clases es uniforme, aunque para la detección de noticias que discuten el titular no se percibe una mejora perceptible.

#### 4.1.3. Discusión

**Representación Bag of Words.** Tras realizar los experimentos se valida que la dimensionalidad es un factor determinante en el rendimiento de los clasificadores. Los mejores resultados se dan en aquellas configuraciones que obtienen representaciones del máximo de dimensiones fijado. En ambos clasificadores las mejores configuraciones son la primera y la tercera, ofreciendo unos resultados muy similares o idénticos para el mismo clasificador. La peor ejecución se corresponde con aquella que aplica unos valores más restrictivos y genera vectores de tamaño menor.

Realizar un filtrado de términos con alta o baja ocurrencia no ha resultado beneficioso generalmente. El filtrado de términos muy frecuentes consigue una leve mejora, pero con unos resultados muy similares respecto a la ejecución sin filtrado.

Se concluye, por tanto, que el clasificador puede caracterizar mejor las clases si analiza el mayor número términos de una muestra en concreto. La aparición de stopwords del dominio o términos poco frecuentes no introducen ruido en la clasificación. En el tratamiento previo del texto ya se realizó un filtrado de stopwords y estandarización de términos, por lo que para este dataset en concreto ya ha supuesto suficiente.

Si se utiliza un filtrado muy restrictivo, los vectores de características tendrán menor dimensionalidad y el clasificador dispondrá de menos datos para generalizar. Es el caso de la quinta configuración, que realizaba un filtrado de aquellos términos con una ocurrencia por debajo del 25 % o superior al 75 %. Además, en este rango se construye el vocabulario a partir de términos de frecuencia intermedia, que no son tan representativos como pueden serlo los más o menos ocurrentes, por lo que da lugar a representaciones poco específicas de cada texto.

Se observa que aumentar el rango mínimo de ocurrencia (min df) influye en gran medida en el tamaño de los vectores. Se observa que tan sólo filtrando el 10 % de los términos menos frecuentes, el tamaño de los vectores se ve reducido a 70, generando una entrada de 140 features para los modelos.

En cuanto al desempeño de los dos clasificadores, vemos que la red neuronal se ve más perjudicada si los datos de entrenamiento tienen una distribución heterogénea en cuanto al número de muestras por clases. Este fenómeno se aprecia mejor en el particionado 2, observando las matrices de confusión, donde se aprecia la tendencia a detectar falsos positivos de la clase mayoritaria. Además, se observa que la clase minoritaria es ignorada por el clasificador. El desempeño de la red neuronal es ligeramente peor con respecto al modelo de random forest, y se resiente en mayor medida por las representaciones con baja dimensionalidad.

Por tanto, parece un buen enfoque aplicar técnicas de oversampling para reducir este sesgo en la clasificación. No obstante, el éxito de aplicar esta técnica se ve muy condicionado por la dimensionalidad de los vectores. El método SMOTE crea nuevas muestras sintéticas a partir de los datos de los que dispone. Si más adelante dichas muestras se ven representadas por vectores de bajas dimensiones, se está introduciendo ruido al clasificador en lugar de mejorar su rendimiento. Este fenómeno se puede observar en los resultados de validación del modelo, en el que se puede ver que la métrica de Recall se reduce considerablemente.

En términos generales, el rendimiento de la representación de Bolsa de términos es aceptable, pero no permite particularizar lo suficientemente bien cada texto para conseguir una precisión por encima del 75 %, en el escenario más realista (partición 2).

**Representación de Word2Vec y Vector de Medias** El rendimiento de esta representación es mejor que el observado en el modelo de Bolsa de términos. Utilizando la representación de vector de medias se obtienen unos valores significativamente mayores. Los modelos de Word2Vec posibilita una caracterización más precisa de los textos.

Los dos clasificadores presentan resultados similares, siendo el modelo de Random Forest el que mejor se comporta en términos sensibilidad y especificidad, aunque no dista mucho del rendimiento de la red neuronal. Por contra, el clasificador basado en Random Forest tiene una tendencia mayor al overfitting. Se puede observar en los resultados del

Aplicar SMOTE sobre el conjunto de entrenamiento, tras generar los vectores de medias ha resultado un acierto, ya que mejora significativamente la precisión

del modelo, y en menor medida el recall. Este resulta el factor más determinante a la hora de mejorar los resultados del clasificador. Gracias a SMOTE se corrige el sesgo en la clasificación observado en los resultados sobre la partición 1. Además, el clasificador consigue identificar muestras de la clase menos representada en el dataset original.

En cuanto al modelo de word2vec más adecuado, los dos obtienen un desempeño adecuado, y son aptos para ser utilizados. El modelo propio obtiene unos tiempos de ejecución menores, con un rendimiento ligeramente superior con respecto al modelo preentrenado oficial. No obstante, sería necesario evaluar si el vocabulario de dicho modelo es lo suficientemente completo como para representar noticias cuando se utilice el clasificador en producción. La terminología de las noticias va variando con el tiempo, en función de los hechos que son tendencia, por lo que si se deseara utilizar en producción un modelo propio sería necesario ir realizando un entrenamiento iterativo de la representación de word2vec, incluyendo nuevas muestras actualizadas. Esta estrategia permite reflejar la nueva terminología, o enriquecer el modelo. Por otra parte, el modelo oficial de word2vec resultaría una alternativa interesante si se prefiere disponer de un modelo estático.

Los resultados obtenidos ofrecen margen para la mejora, que se conseguirá optimizando la red neuronal utilizada.

## **4.2. Clasificadores basados en Redes Neuronales Profundas**

Una vez estudiados los modelos de representación convenientes, realizamos la experimentación del clasificador de noticias. Para ello, se han desarrollado modelos basados en la arquitectura de red neuronal hacia delante.

Para el desarrollo de la red neuronal hacia delante se partirá del modelo de perceptrón multicapa de la experimentación de modelos de representación. La representación que se utilizará será la de word2vec aplicando vector de medias, en la que se obtuvo unos resultados mejores, en términos de sensibilidad y especificidad.

### **4.2.1. Modelo de Red Neuronal Hacia Delante**

El objetivo de esta validación es analizar los parámetros clásicos de un modelo de red de neuronas y dar con la configuración adecuada. Se quiere comprobar si la arquitectura de red neuronal hacia delante tiene unos resultados adecuados en el análisis de posicionamiento de noticias, o si por el contrario resultaría mejor hacer uso de arquitecturas más complejas.

La complejidad de desarrollar una arquitectura óptima reside en la gran flexibilidad que nos ofrecen los modelos de redes de neuronas. Los hiperparámetros de la red se pueden dividir en tres grupos: aquellos que son propios de la red (número de neuronas por capa, número de capas), funciones de activación y normalización, y parámetros de entrenamiento y aprendizaje (tamaño del batch, tasa de aprendizaje, número de epochs).

**Diagnóstico de la configuración inicial.** Antes de comenzar con la experimentación, es importante realizar un diagnóstico de la configuración por defecto que se utilizó en la validación de los modelos de representación. En base a los resultados detectaremos si nos encontramos ante un escenario de *overfitting* o *underfitting*, y orientaremos la experimentación aplicando configuraciones que subsanen uno de estos problemas.

A la hora de identificar el error de un modelo, se habla del **bias** y la **varianza**. La primera variable mide el error del modelo con respecto a las muestras de entrenamiento, mientras que la segunda representa la variabilidad del modelo a la hora de aproximarse a los datos de entrada (grados de libertad del modelo). Si el modelo se encuentra en un escenario de *overfitting*, quiere decir que presenta una alta varianza, debido a que la representación tiene más parámetros de los necesarios para que el modelo generalice correctamente. Esto provoca que el modelo se ajuste en gran medida a los datos de entrenamiento, pero presente una efectividad significativamente menor sobre el conjunto de entrenamiento. Para paliar este efecto se suelen utilizar técnicas de regularización, para limitar los grados de libertad del modelo en la fase de entrenamiento.

Por el contrario, un escenario de *underfitting* presenta un bias alto, ya que el modelo resultante no es capaz de ajustarse lo suficientemente bien a los datos de entrenamiento o validación. Para mejorar el desempeño del modelo se puede considerar una arquitectura más compleja, aumentando el número de capas y neuronas por capa, o ajustar parámetros propios del entrenamiento como el número de epochs, tamaño del batch o tasa de aprendizaje.

La configuración base del clasificador que se utilizó es la que se expone a continuación:

- **Número de capas ocultas:** 2.
- **Neuronas de la capa oculta:** 300 para la primera, 100 para la segunda.
- **Función de activación:** ReLU.
- **Tamaño del batch:** 50 muestras.
- **Estrategia de batch:** Aprendizaje con Mini-batch.
- **Tasa de aprendizaje:** 0.01 (constante).
- **Número de epochs:** 20.
- **Función de coste:** Gradiente Descendiente.
- **Normalización de la capa de salida:** Entropía cruzada con logits.

Para el análisis partimos de los resultados del mejor escenario reportado, la ejecución del clasificador utilizando el modelo de vector de medias de *embeddings* de *word2vec*. Se estudian los resultados sobre el particionado 2, aplicando el método de oversampling SMOTE (página 59).

En los resultados de dicha experimentación se reportan unos valores de *accuracy* en torno al 89 %, para el conjunto de entrenamiento, y un 79 %, para el conjunto de validación. Se detecta *overfitting* , por la diferencia entre los resultados de entrenamiento y validación. Visualizando la función de coste de la ejecución con el modelo propio y con el modelo entrenado oficial, se puede apreciar que esta última presenta un *overfitting* mayor.

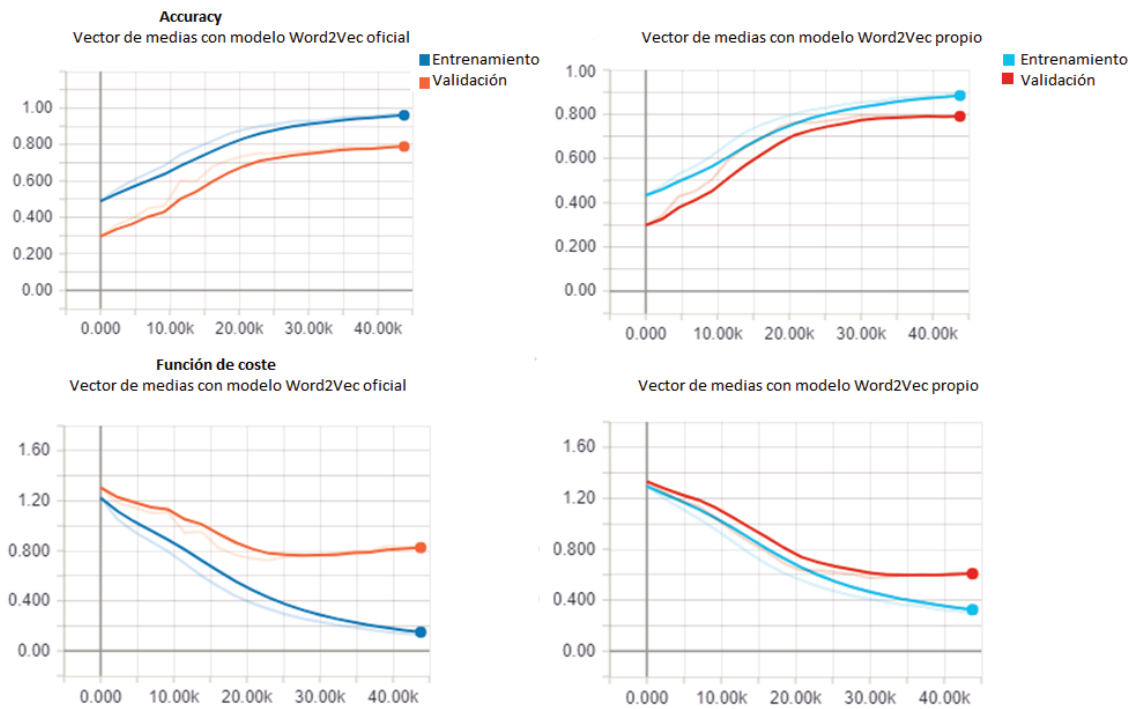


Figura 30: Función de Accuracy y de coste para la configuración inicial, sobre el conjunto de entrenamiento/validación, particionado 2

En cuanto a los resultados sobre el conjunto de test, se reportó una diferencia en las métricas de un 5 % y unos resultados en las métricas mayores.

Por tanto, para mejorar la efectividad del modelo, es necesario tratar este overfitting, que si bien es leve, repercute en la efectividad del modelo. Para el tratamiento del overfitting se han considerado tres métodos de regularización, en orden creciente de complejidad: *Early Stopping*, *Dropout* y Regresión de Ridge (L2). El objetivo de los métodos de regularización es limitar los grados de libertad del modelo (disminuir la varianza), con el fin de evitar el sobreajuste a los datos de entrenamiento.

Otra posibilidad es la de experimentar con arquitecturas más complejas, con mayor número de capas o neuronas por capa, para intentar mejorar los valores en las métricas. Puesto que una arquitectura más compleja es más susceptible de presentar overfitting, se podría combinar con alguno de los métodos de regularización contemplados. Para la experimentación final se opta por trabajar con la arquitectura de dos y tres capas ocultas.

En la fase de experimentación se pretenden validar qué método de regularización

resulta más efectivo, cuál es el método de aprendizaje más óptimo, y si se puede obtener una arquitectura de red que reporte una efectividad mayor.

Las configuraciones de la experimentación final se engloban en tres grupos: arquitecturas, parámetros aprendizaje y métodos de regularización. Durante la experimentación se ha seguido el mecanismo a continuación descrito. En primer lugar se han planteado las arquitecturas de red a considerar (número de capas y número de neuronas por capa), tras una experimentación previa. Una vez definidas las arquitecturas, se ha realizado un estudio de las diversas configuraciones posibles sobre los parámetros de aprendizaje (función de activación, función de optimización, tasa de aprendizaje).

**Arquitecturas de red.** En base a los resultados de partida comentados, se ha optado con experimentar con arquitecturas de dos y tres capas ocultas.

- Dos capas ocultas, con 300 y 100 neuronas respectivamente.
- Tres capas ocultas, con 300, 150 y 100 neuronas respectivamente.

El propósito de experimentar con varias arquitecturas es determinar qué arquitectura de las contempladas es más óptima en términos de tiempo y de efectividad del modelo resultante.

**Funciones de optimización y tasa de aprendizaje.** Se ha experimentado con dos métodos de optimización distintos: **Gradiente Descendiente** y **Gradiente Descendiente con momentum**.

El método del gradiente descendiente calcula la dirección del gradiente que minimiza la función de pérdida. Este método toma como parámetro la tasa de aprendizaje.

En este análisis se ha considerado un esquema de aprendizaje estático y varios dinámicos, con el propósito de determinar cuál de las dos estrategias resulta más provechosa. Para actualizar la tasa de aprendizaje disponemos de múltiples opciones, por un lado podemos emplear un decrecimiento exponencial o decrecimiento fijo, y por otro lado, podemos optar por aplicar la actualización de forma incondicional o determinada por la tasa de aciertos de entrenamiento y validación en la *epoch* o iteraciones anteriores.

El esquema de actualización escogido consiste en actualizar el learning rate en cada iteración, independientemente de los resultados, y con un patrón de actualización exponencial. Para ello se ha empleado el método *tf.train.exponential\_decay* de TensorFlow, que actualiza la tasa de aprendizaje según la siguiente expresión [61]:

$$LR(iteración) = LR(iteración - 1) * factorBase^{iteración/NitercionesEpoch}$$

Figura 31: Patrón de actualización de la tasa de aprendizaje.

Seguidamente se detallan los esquemas de actualización de learning rate que se han seguido para el optimizador clásico de gradiente descendiente:



- Learning rate constante de 0.01.
- Learning rate dinámico con valor de partida 0.01 y factor base 0.95.
- Learning rate dinámico con valor de partida de 0.01 y factor base 0.90.
- Learning rate dinámico con valor de partida 0.05 y factor base 0.95.
- Learning rate dinámico con valor de partida 0.05 y factor base 0.90.

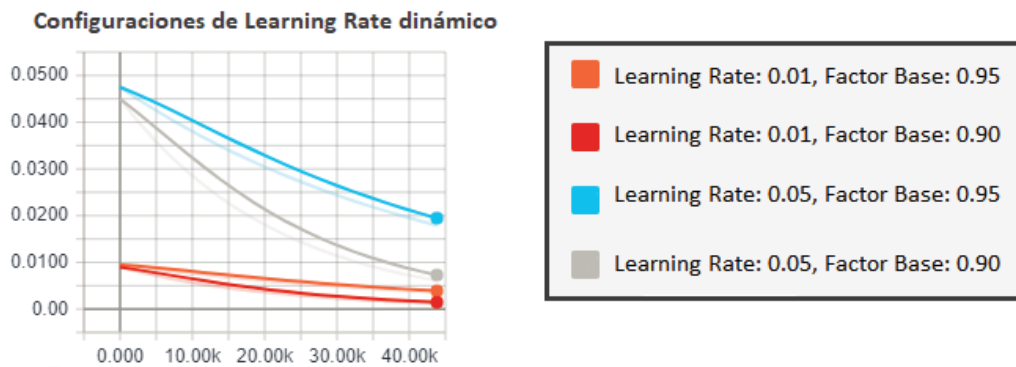


Figura 32: Configuraciones de learning rate dinámico utilizadas en la experimentación.

El segundo optimizador considerado es el método momentum, similar en planteamiento al método del Gradiente Descendiente, pero incorporando la noción de aceleración (momento), calculando el gradiente actual teniendo en cuenta los anteriores. Este algoritmo tiene un parámetro extra, el momentum, que en esta experimentación se ha utilizado en su configuración por defecto. Para este optimizador se ha empleado un learning rate estático con dos configuraciones: 0.001 y 0.005.

Sobre el mejor optimizador y learning rate obtenido se aplicarán las estrategias de regularización para mejorar los resultados del modelo y dar con una configuración final de la arquitectura en particular.

La función permite configurar para aplicar el momentum clásico o el método de Nesterov. En esta experimentación se ha optado por aplicar el método clásico para el modelo de dos capas y el modelo de Nesterov para el modelo de tres capas.

**Métodos de Regularización: Early Stopping.** Para la implementación de Early Stopping (Sección 2.4.2) se fija un valor de *paciencia* configurable. Este parámetro determina el número de iteraciones máximo seguidos en el que se permite que el modelo no experimente una mejora en el valor de coste. El valor de *paciencia* es un entero que se multiplica por el número de iteraciones por epoch. El número de iteraciones por epoch viene dado por el tamaño de los datos de entrenamiento (115308 muestras tras aplicar SMOTE) y el tamaño del batch (fijado en 50). Para el fichero de entrenamiento utilizado el número de iteraciones por epoch es de 2036.

Cada veinte iteraciones se revisa el valor de la función de coste en dicho momento, y se compara con el menor resultado registrado hasta el momento. Si se percibe una mejora, se almacena el valor y se guarda en disco la versión actual del clasificador. Si, por el contrario, no se ha obtenido una mejor, el contador de estacionalidad se incrementa en uno. Al final de cada epoch se comprueba la estacionalidad del modelo, y si es mayor que el ratio de paciencia fijado, se interrumpe el entrenamiento. En el disco permanece almacenado el último guardado del modelo.

En caso de no llegar a dicha estacionalidad, el modelo entrenaría un número máximo de epochs, configurado en veinte, como en la configuración base. No obstante, en los resultados se percibe que el modelo comienza a diverger antes de dicho punto, por lo que en la práctica el modelo no llegará a entrenar el máximo de epochs fijado.

Tensorflow facilita la creación de puntos de guardado o checkpoints, por medio de la creación de nodos de guarda o *saver* [62]. Para el guardado de la versión actual del modelo se utiliza el método *save*, especificando la ruta de almacenamiento.

En la experimentación se probará con algunos valores de paciencia, para obtener cuál de ellos es el más prudente y reporta unos valores mejores. Se experimenta con valores superiores a uno para poder observar la estacionalidad un número representativo de iteraciones. El objetivo de la validación es observar en qué medida se puede optimizar el modelo, sin modificar la arquitectura, ni aplicando transformaciones en la fase de entrenamiento.

- Early stopping con paciencia 1,5. Se permite una estacionalidad máxima de 3459 iteraciones.
- Early stopping con paciencia 2. Se permite una estacionalidad máxima de 4612 iteraciones.
- Early stopping con paciencia 3. Se permite una estacionalidad máxima de 6918 iteraciones.

**Métodos de Regularización: Dropout.** En *Tensorflow* esta técnica se abstraee mediante una capa especial que se encarga de realizar el filtrado y que se sitúa tras cada oculta. Dicha capa se crea mediante la función de *dropout* de la librería *nn* [61]. La capa se configura por medio del parámetro *keep prob*, que expresa la probabilidad de permanencia de una neurona en concreto. El filtrado sobre cada neurona se realiza de forma independiente, por lo que la permanencia o descarte de una no condiciona el resto.

Durante la fase de entrenamiento, se configura este parámetro en la probabilidad deseada. En la fase de testeo se fija este parámetro a uno para no activar el *dropout*, y así realizar las predicciones con la red original, que ha aprendido por medio de esta técnica.

La arquitectura de la red con *dropout* es la que se expone a continuación:

Tras una experimentación previa se ha optado por utilizar una tasa de dropout por debajo del 50 %. También se observó que la aplicación de esta técnica ralentizaba

```

with tf.name_scope("FNN"):
    # Definimos las capas ocultas
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1", activation=activation)
    dropout1 = tf.nn.dropout(hidden1, keep_prob, name="dropout_1_out")
    hidden2 = tf.layers.dense(dropout1, n_hidden2, name="hidden2", activation=activation)

```

Figura 33: Código de definición de la capa de dropout entre dos capas ocultas

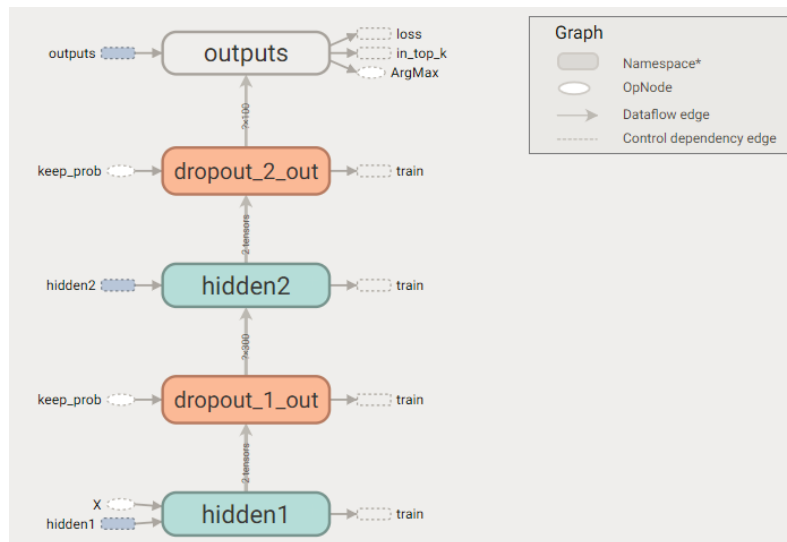


Figura 34: Arquitectura de la red con las capas de dropout

el entrenamiento, por lo que se optó por aumentar la tasa de aprendizaje de 0.01 a 0.05.

Las configuraciones que se validarán en esta experimentación son las siguientes:

- Tasa de dropout del 25 %.
- Tasa de dropout del 35 %.
- Tasa de dropout del 50 %.

**Métodos de Regularización: Regresión de Ridge (L2).** Para aplicar la regularización es necesario hacer dos modificaciones al modelo. En primer lugar, se especifica en la definición de la capa oculta el método regularizador que se aplicará sobre las neuronas. La capa oculta se define mediante el método *layers.dense*, y el regularizador se configura mediante el argumento *kernel regularizer*. Una neurona artificial se compone internamente por dos variables: el peso (o *kernel*, en *Tensorflow*) y el *bias*. La regularización se aplicará sobre el peso. La regularización L2 se encuentra implementada por medio del método *l2 regularizer* de *Tensorflow* [61].

Durante el entrenamiento, TensorFlow agregará las penalizaciones agregadas de cada capa a la colección *regularization losses*, por lo que la segunda modificación

```

with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
    if l2_scale >= 0.0:
        reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
        loss = tf.reduce_mean(xentropy, name="base_loss")
        final_loss = tf.add_n([loss] + reg_losses, name="loss")

```

Figura 35: Código del cálculo de la función de coste con el factor de penalización

consiste en obtener dichas penalizaciones y sumarlas al valor de coste obtenido en la iteración actual.

El regularizador tiene un parámetro para fijar la escala de la regularización,  $\alpha$ . Para obtener la escala adecuada se ha experimentado con varios valores hasta dar con el rango adecuado. En la experimentación final se ha ejecutado el modelo con las siguientes configuraciones:

- Escala de regularización de 0.001.
- Escala de regularización de 0.002.
- Escala de regularización de 0.005.

**Experimentación. Metodología.** El proceso de obtener la mejor configuración se ha llevado a cabo mediante una optimización manual. Para la experimentación se ha convertido el clasificador base en un clasificador parametrizable, con los siguientes hiperparámetros configurables:

- *Activation function*: Función de activación de las capas ocultas. Valores posibles: “relu”, “leaky relu” y “relu”. Valor por defecto: “relu”. Para la definición de las funciones de activación se utiliza la implementación de TensorFlow para las funciones de ReLU, Leaky ReLU y ELU [61].
- *Epochs*: Número de epochs. Valor por defecto: 20.
- *Hidden neurons*: Array de neuronas por capa. Soporta hasta 5 capas ocultas. Arquitectura por defecto: [300, 100]
- *Batch size*: Tamaño del batch. Valor por defecto: 50.
- *Learning rate*: Tasa de aprendizaje. Valor por defecto: 0.01.
- *Learning decrease*: Habilita/deshabilita la actualización dinámica de la tasa de aprendizaje. Valor por defecto: 1 (deshabilitado)
- *Early stopping*: Habilita/deshabilita el método de Early Stopping. Valor por defecto: False (deshabilitado).
- *Early stopping patience*: Número de iteraciones máximo. Valor por defecto: 2 (se aplica si está activado).

- *Dropout rate*: Porcentaje de dropout. Valor por defecto: 0.0 (No se aplica).
- *L2 scale*: Escala de regularización L2. Valor por defecto: 0.0 (No se aplica regularización).
- *Optimizer*. Optimizador utilizado, configurable como gradiente descendiente o momentum. Valor por defecto: Gradiente Descendiente.

El clasificador hace uso de funciones de registro de escalares, para poder generar las gráficas de accuracy y coste en Tensorboard. Para facilitar la obtención de

En base a estos hiperparámetros se ha llevado una experimentación previa, con el fin de explorar cuáles resultaban más interesantes. Se experimentó con varias arquitecturas de red, funciones de activación y regularización.

En estas pruebas previas se experimentó con arquitecturas de hasta 5 capas ocultas, con varias combinaciones en el número de neuronas. Sin embargo, el aumento de capas traía consigo un aumento del overfitting, y el clasificador dejaba de ser óptimo a partir de la cuarta capa. Finalmente se optó por centrar la experimentación en una única arquitectura, la inicialmente utilizada, y tratar de subsanar el overfitting aplicando métodos de regularización.

Para la validación y testeo de las diversas configuraciones se ha llevado a cabo un proceso de validación simple, utilizando el particionado 2, presentado en la experimentación con métodos de representación.

Durante la fase de experimentación ha resultado de gran ayuda la herramienta *Tensorboard* de *Tensorflow*, que permite el exportado de escalares y vectores para la creación de gráficas e histogramas, que se pueden visualizar en tiempo de ejecución. Las gráficas de accuracy y de coste que se reportan se han obtenido a partir de esta herramienta.

La obtención de las métricas de desempeño se realiza de la misma forma. Para el cálculo de la métrica de accuracy se utiliza la función de accuracy de *Tensorflow*, y para las métricas de precisión y recall, junto con la matriz de confusión, se obtienen utilizando los métodos proporcionados por *Scikit Learn*.

La versión del modelo de vector average que se emplea es la del modelo propio, aplicando el método de oversampling SMOTE sobre la partición de entrenamiento, la que obtuvo unos resultados mejores en la fase de experimentación de modelos de representación de textos.

#### 4.2.2. Resultados

**Arquitectura de dos capas ocultas.** En primer lugar se ha realizado una experimentación previa sobre distintos optimizadores y tasas de aprendizaje. La ejecución con el optimizador momentum han reportado resultados mejores en términos de especificidad y sensibilidad. En cuanto a los tiempos de ejecución, se reportan resultados similares, no se aprecia una disminución sustancial sobre aplicar un optimizador u otro.

Las mejores configuraciones del optimizador del Gradiente Descendiente se corresponden con aquellas que definen una tasa de aprendizaje dinámico, con un factor

base de actualización en el rango de 0.9 a 0.95, partiendo de una tasa de aprendizaje de 0.05. Las dos ejecuciones del optimizador momentum ofrecen buenos resultados en el valor predictivo positivo y especificidad del modelo.

Optimizador	Learning Rate	factor base	epochs	T. aciertos (E)	T. aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	T. ejecución (s)
GD	0.01	1	20	0.97	0.79	0.97	0.8	0.97	0.79	328.18
GD	0.01	0.95	20	0.92	0.75	0.92	0.77	0.92	0.75	323.72
GD	0.01	0.9	20	0.84	0.68	0.84	0.76	0.84	0.68	321.96
GD	0.05	0.95	20	1.0	0.82	1.0	0.81	1.0	0.82	320.71
GD	0.05	0.9	20	1.0	0.81	1.0	0.8	1.0	0.81	324.52
momentum	0.005	1	20	1.0	0.81	1.0	0.81	1.0	0.81	329.24
momentum	0.001	1	20	0.97	0.78	0.97	0.79	0.97	0.78	328.91

Cuadro 9: Resultados de diversas estrategias de aprendizaje, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas.

Para escoger la mejor configuración del optimizador se han obtenido las curvas de aprendizaje y tasa de aciertos de las dos mejores configuraciones con Gradiente Descendiente clásico y momentum. Se puede observar que pese a reportar métricas de tasa de aciertos, valor predictivo positivo y sensibilidad similares, los patrones de aprendizaje son similares. En las configuraciones de Gradiente Descendiente clásico se identifica una divergencia del modelo, lo que provoca que la tasa de aciertos sobre el conjunto de validación alcance una cota máxima en torno al 80 %. La mejor configuración reportada es aquella que emplea el optimizador momentum con una tasa de aprendizaje de 0.001, por lo que se empleará en la siguiente fase de la experimentación.

Sobre esta configuración se ejecutan tres regularizaciones distintas. Con la aplicación de Early Stopping, con una paciencia de 3 epochs, se consigue un valor predictivo positivo en torno al 80 % y especificidad de 76 %, consiguiendo una reducción de un 23 % sobre el tiempo de ejecución del modelo.

Optimizador	Tasa de aprendizaje	epochs	T.aciertos (E)	T.aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	Paciencia	T.ejecución (s)
momentum	0.001	13	0.93	0.75	0.93	0.79	0.93	0.75	2	217.49
momentum	0.001	12	0.9	0.75	0.91	0.77	0.9	0.75	1.5	202.09
momentum	0.001	15	0.94	0.76	0.94	0.79	0.94	0.76	3	253.74

Cuadro 10: Resultados de aplicación de técnicas de early stopping, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas.

La aplicación de Dropout da unos resultados desequilibrados de especificidad-sensibilidad en la primera configuración, en la que se establece una probabilidad de descarte por neurona del 25 %. La mejor configuración es aquella que aplica una probabilidad del 50 %, reportando unos resultados equilibrados, y con un valor predictivo positivo de 0.8. La aplicación de la técnica de Dropout supone una penalización del 20 % en el tiempo de ejecución del modelo.

Optimizador	Tasa de aprendizaje	epochs	T.aciertos (E)	T.aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	Tasa Dropout	T.ejecución (s)
momentum	0.001	20	0.77	0.67	0.78	0.77	0.78	0.67	0.25	401.17
momentum	0.001	20	0.85	0.76	0.85	0.78	0.85	0.76	0.35	404.59
momentum	0.001	20	0.9	0.79	0.9	0.8	0.9	0.79	0.5	401.71

Cuadro 11: Resultados de aplicación de la técnica de Dropout, sobre el conjunto de entrenamiento/validación, arquitectura de 2 capas.

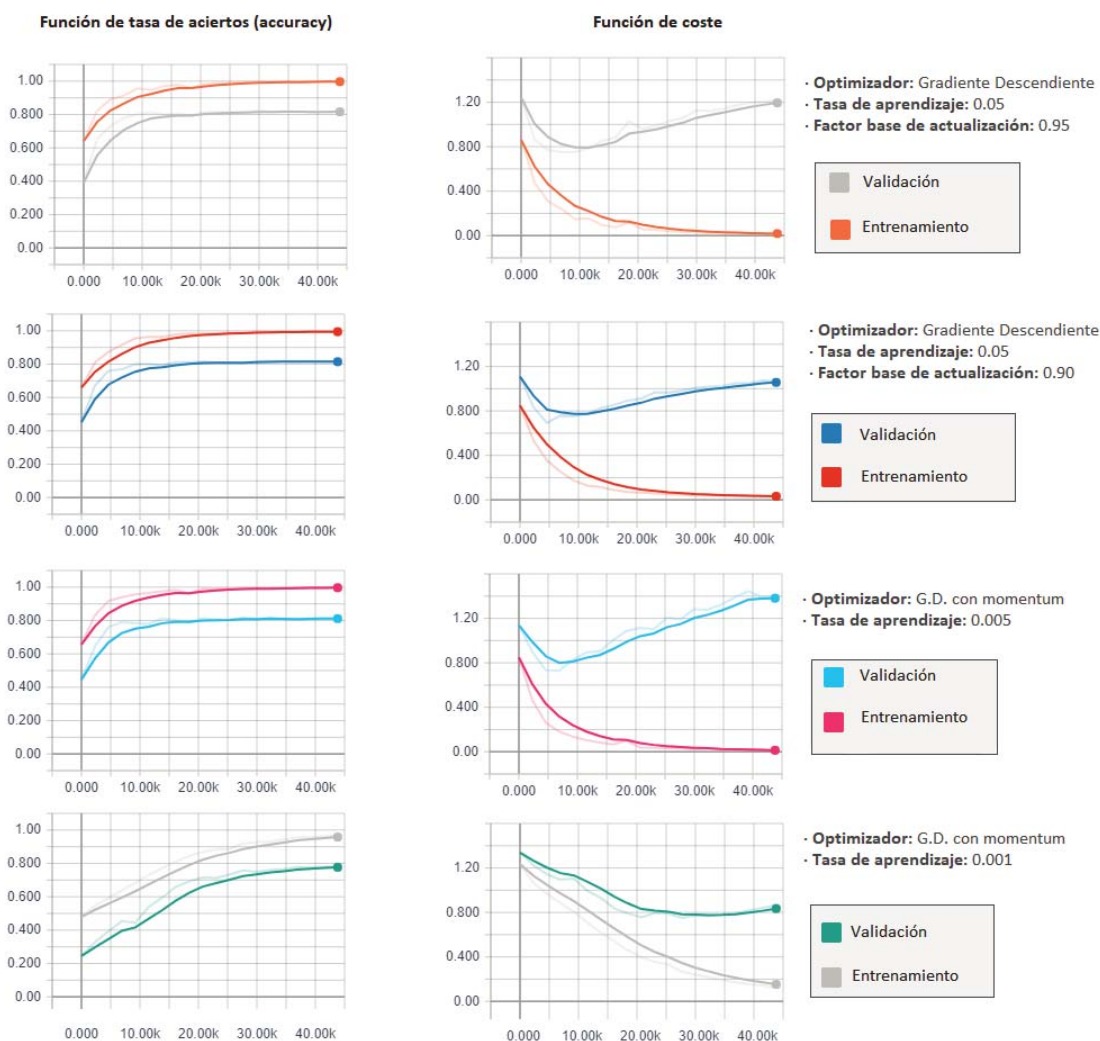


Figura 36: Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, sobre el conjunto de entrenamiento/validación.

El último experimento con regularizaciones se corresponde con la aplicación de regularización de Ridge. El mejor valor reportado (regularización de Ridge con escala 0.001) consigue un valor predictivo positivo del 80 %.

Optimizador	Tasa de aprendizaje	epochs	T.aciertos (E)	T.aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (V)	Escala de regularización	T.ejecución (s)
momentum	0.001	20	0.96	0.79	0.96	0.8	0.96	0.79	0.001	344.52
momentum	0.001	20	0.93	0.78	0.94	0.8	0.93	0.78	0.002	351.61
momentum	0.001	20	0.87	0.75	0.87	0.79	0.87	0.75	0.005	345.84

Cuadro 12: Resultados de aplicación de la técnica de regularización de Ridge, sobre el conjunto de validación/test, arquitectura de 2 capas.

De nuevo observamos las funciones de coste y de tasa de aciertos para comparar la efectividad de las distintas técnicas de regularización. se escoge la mejor configuración de cada grupo y se comparan las funciones de coste. La configuración que consigue regularizar mejor el modelo, manteniendo valores cercanos en la función de



coste entre la partición de entrenamiento y la de test es el modelo de Dropout. Se observa una mejora perceptible del overfitting y convergencia de la aproximación del conjunto de validación, si lo comparamos con respecto a la configuración base (página 66). Recordamos que la experimentación se ha realizado utilizando el modelo de word2vec pre-entrenado oficial.

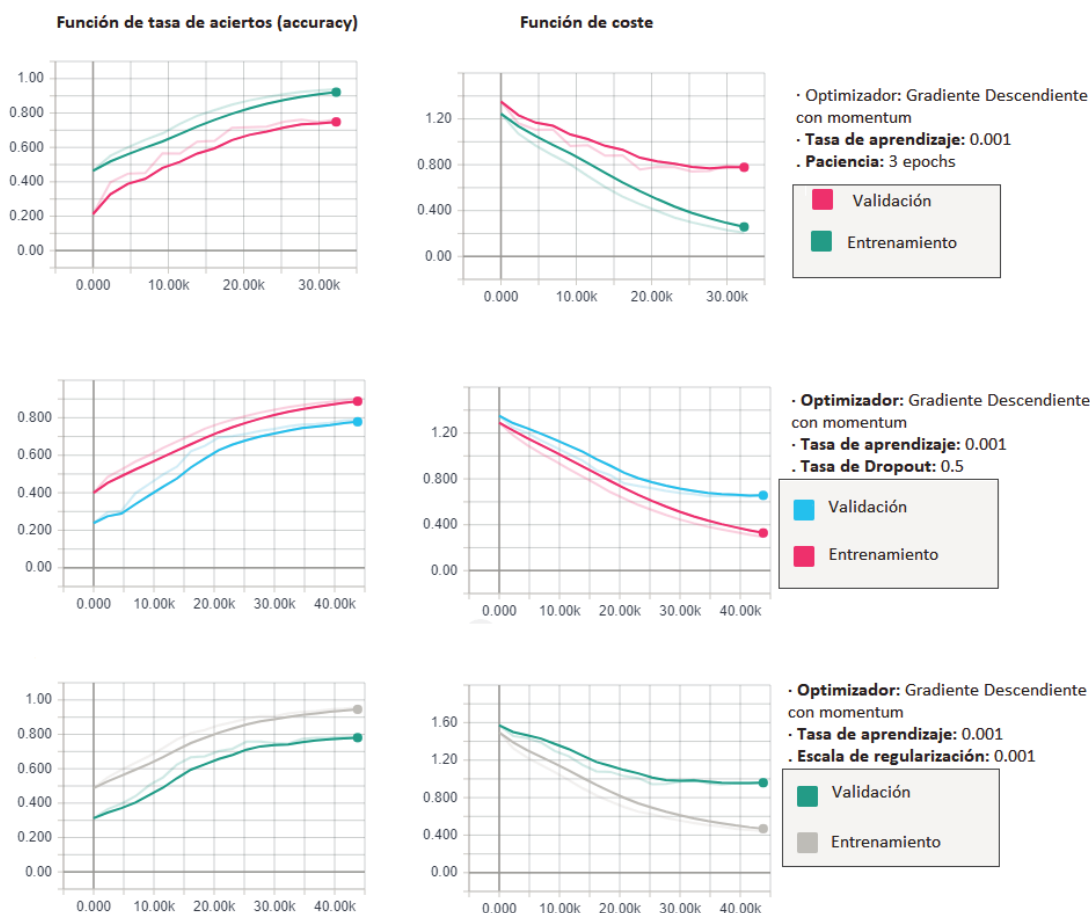


Figura 37: Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, sobre el conjunto de entrenamiento/validación.

**Configuración final del modelo de dos capas.** Finalmente ejecutamos la mejor configuración sobre el conjunto de test, aquella correspondiente a la aplicación de dropout con una probabilidad de descarte de 0.50.

Optimizador	tasa de aprendizaje	epochs	T. aciertos (T)	Precisión (T)	Recall (T)	Tasa de dropout	T. ejecución (s)
momentum	0.001	20	0.82	0.85	0.82	0.5	468.30

Cuadro 13: Resultados de la configuración final, sobre el conjunto de test, arquitectura de 2 capas.

Las curvas ROC nos permiten determinar la efectividad del clasificador sobre cada clase, en términos de precisión y *recall*. Se observa una mejora en el valor de



AUC en todas las clases, comparadas con los resultados de la red neuronal antes de la optimización (página 62). Mejora la caracterización de todas las clases, excepto la correspondiente a noticias no relacionadas, que mantiene un comportamiento similar.

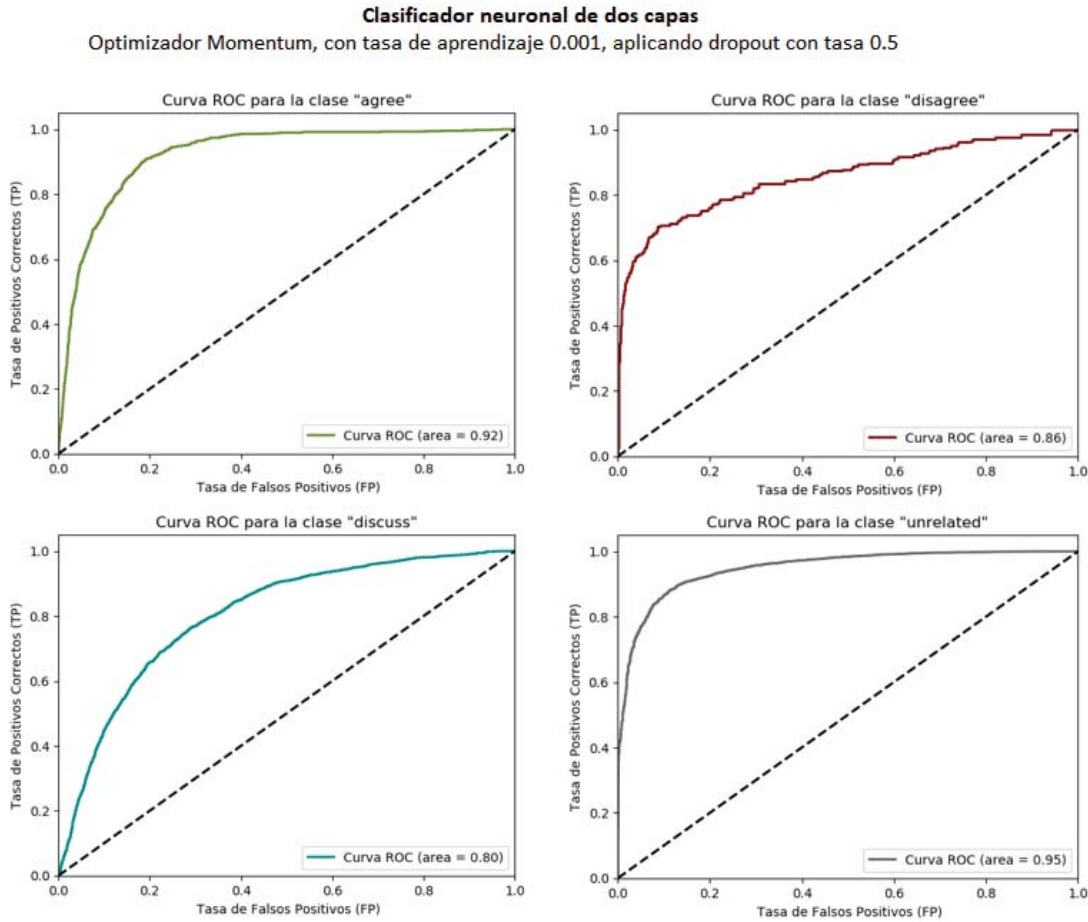


Figura 38: Curvas ROC de la configuración final sobre el conjunto de test

**Arquitectura de tres capas ocultas.** La experimentación de métodos de optimización es similar a la realizada con el modelo de dos capas ocultas, utilizando en este caso la variante Nesterov del método Momentum [58]. De partida se observa un incremento de 90 segundos en el tiempo de ejecución del modelo, con respecto al modelo de 2 capas en el mismo grupo de configuraciones. Se observan configuraciones que consiguen aumentar la precisión del modelo en términos de tasa de aciertos y recall, ofreciendo un valor reportado de 0.82. Además, se puede ver que la configuración base (tasa de aprendizaje estática de 0.01) de por sí ofrece unos resultados mejores, sobre la misma configuración sobre el modelo de dos capas (página 73).

Tomamos las dos mejores configuraciones del optimizador de Gradiente Descendiente y Momentum de Nesterov, en base a su tasa de aciertos, valor predictivo positivo y recall. Las gráficas de coste nos ofrecen el comportamiento del modelo

Optimizador	T. aprendizaje	factor base	epochs	T. aciertos (E)	T. aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (T)	T. ejecución (s)
Gradiente Descendiente	0.01	1	20	0.99	0.81	0.99	0.79	0.99	0.81	414.76
Gradiente Descendiente	0.01	0.95	20	0.99	0.8	0.99	0.79	0.99	0.8	415.61
Gradiente Descendiente	0.01	0.9	20	0.96	0.77	0.96	0.77	0.96	0.77	416.99
Gradiente Descendiente	0.05	0.95	20	1.0	0.81	1.0	0.8	1.0	0.81	420.35
Gradiente Descendiente	0.05	0.9	20	1.0	0.82	1.0	0.81	1.0	0.82	417.22
Gradiente Descendiente	0.05	0.95	20	1.0	0.82	1.0	0.81	1.0	0.82	417.01
Gradiente Descendiente	0.05	0.9	20	1.0	0.81	1.0	0.8	1.0	0.81	379.62
Momentum Nesterov	0.005	1	20	1.0	0.82	1.0	0.8	1.0	0.82	423.93
Momentum Nesterov	0.001	1	20	1.0	0.79	1.0	0.79	1.0	0.79	422.97

Cuadro 14: Resultados de distintas configuraciones sobre optimizadores y tasa de aprendizaje, arquitectura de 3 capas, conjunto de entrenamiento y validación.

a lo largo del entrenamiento, se concluye que las configuraciones más óptimas para el escenario de configuración considerado son la que aplica Gradiente descendiente con tasa de aprendizaje estática y Gradiente Descendiente Momentum de Nesterov, con una tasa de aprendizaje de 0.001. Nos decantamos por utilizar la segunda de las expuestas y aplicar regularizaciones sobre la misma.

El primer grupo de regularizaciones, correspondientes a la aplicación de Early Stopping, consiguen reducir los tiempos de ejecución a la mitad, aunque degradan ligeramente la calidad del modelo. De nuevo se observa una ligera reducción en los tiempos con respecto al modelo de dos capas.

Optimizador	T. aprendizaje	factor base	epochs	T. aciertos (E)	T. aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (T)	Paciencia	T. ejecución (s)
Momentum Nesterov	0.001	1	9	0.95	0.76	0.95	0.77	0.95	0.76	2	191.21
Momentum Nesterov	0.001	1	9	0.96	0.77	0.96	0.78	0.96	0.77	1.5	191.82
Momentum Nesterov	0.001	1	10	0.96	0.77	0.96	0.78	0.96	0.77	3	214.674

Cuadro 15: Resultados de distintas configuraciones aplicando técnica de Early Stopping, arquitectura de 3 capas, conjunto de entrenamiento y validación.

En el apartado de regularizaciones de Dropout se consigue unas métricas de 81 % en tasa de aciertos, valor predictivo positivo y especificidad. La mejor configuración se corresponde con una aplicación de tasa de Dropout de 0.50. Se observa que aplicar una tasa de dropout baja (0.25) degrada perceptiblemente el desempeño del modelo.

Optimizador	T. aprendizaje	factor base	epochs	T. aciertos (E)	T. aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (T)	Tasa dropout	T. ejecución (s)
Momentum Nesterov	0.001	1	20	0.73	0.59	0.73	0.75	0.73	0.59	0.25	522.98
Momentum Nesterov	0.001	1	20	0.84	0.75	0.84	0.78	0.84	0.75	0.35	519.29
Momentum Nesterov	0.001	1	20	0.93	0.81	0.93	0.8	0.93	0.81	0.5	524.27

Cuadro 16: Resultados de distintas configuraciones aplicando regularización Dropout, arquitectura de 3 capas, conjunto de entrenamiento y validación.

Finalmente se experimenta con regularizaciones de L2, obteniendo en la mejor configuración un resultado de 0.8 en las métricas objetivas medidas.

Optimizador	T. aprendizaje	factor base	epochs	T. aciertos (E)	T. aciertos (V)	Precisión (E)	Precisión (V)	Recall (E)	Recall (T)	Escala de regularización	T. ejecución (s)
Momentum Nesterov	0.001	1	20	0.99	0.8	0.99	0.8	0.99	0.8	0.001	441.92
Momentum Nesterov	0.001	1	20	0.99	0.79	0.99	0.8	0.99	0.79	0.002	440.36
Momentum Nesterov	0.001	1	20	0.96	0.76	0.96	0.79	0.96	0.76	0.005	439.42

Cuadro 17: Resultados de distintas configuraciones aplicando técnica de Early Stopping, arquitectura de 3 capas, conjunto de entrenamiento y validación.

Finalmente se visualizan las funciones de accuracy y coste, y se determina que la configuración más óptima se corresponde con la aplicación de regularización L2, con escala 0.001.



Figura 39: Comparación de las diversas configuraciones sobre optimizadores y tasa de aprendizaje, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación

**Configuración final del modelo de tres capas.** A continuación se aportan las métricas finales sobre el modelo de tres capas.

Optimizador	T. aprendizaje	factor base	epochs	T. aciertos (V)	Precisión (T)	Recall (T)	Escala L2	T. ejecución (s)
Momentum Nesterov	0.001	1	20	0.86	0.87	0.86	0.001	428.05

Cuadro 18: Resultados de sobre el conjunto de test de la configuración final, arquitectura de 3 capas.

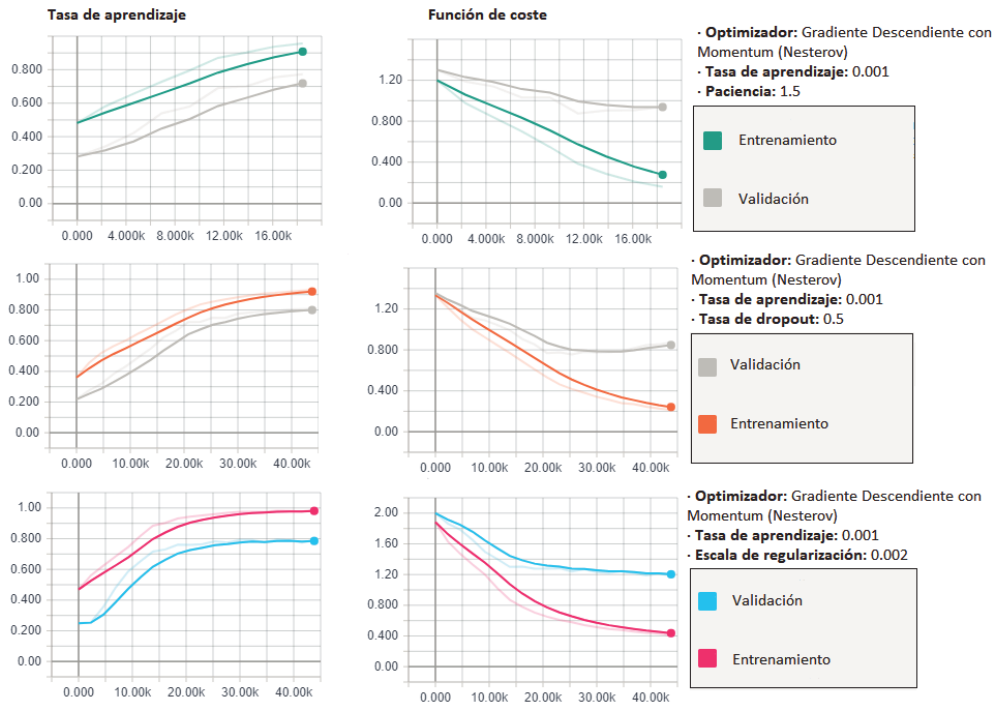


Figura 40: Comparación de las diversas configuraciones sobre técnicas de regularización, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación

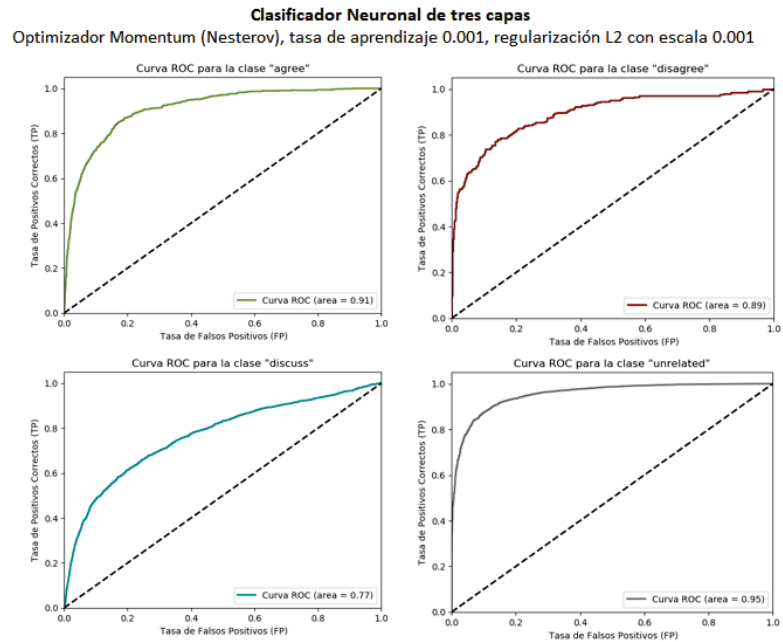


Figura 41: Comparación de las diversas configuraciones sobre técnicas de regularización, arquitectura de tres capas, sobre el conjunto de entrenamiento/validación

### 4.2.3. Discusión

De las dos arquitecturas ofrecidas, la que presenta una efectividad mayor es el modelo de 3 capas, con regularizador momentum (variante Nesterov) y regularización L2 con escala 0.001. Con esta configuración se consigue unos resultados en torno al 86 %, en valor predictivo positivo, tasa de aciertos y especificidad. Se valida que el aumento de capas mejora la capacidad de clasificación del modelo, teniendo en cuenta que este incremento tiene que aplicarse cuidadosamente, sobretodo en escenarios de overfitting.

A raíz de los resultados se constata que la combinación de ajuste de optimizadores y tasa de aprendizaje, junto con la aplicación de técnicas de regularización consigue mejorar la convergencia del modelo. En el apartado de optimización la mejor configuración se consigue reduciendo la tasa de aprendizaje y aplicando la versión momentum del Gradiente Descendiente. También se ha observado en ambos modelos que con determinadas configuraciones con tasa de aprendizaje dinámica se consiguen buenos resultados, pero son configuraciones que requieren un ajuste más cuidadoso y no reportan una mejora significativa. El método momentum se configura con una tasa de aprendizaje de un orden inferior, sin llevar a un impacto significativo en el tiempo de ejecución, gracias a la estrategia en la que basa su optimización (factor momento).

La estrategia de actualización dinámica de la tasa de aprendizaje ha provocado una divergencia mayor del coste de validación con respecto al coste de entrenamiento (página 74). Esto provoca un comportamiento muy similar en la tasa de aciertos reportadas, en la que se puede visualizar una estacionalidad en los resultados. A partir de cierto punto el modelo sigue ganando conocimiento sobre los datos de entrenamiento, pero se sobreajusta a los mismos y eso da lugar a una capacidad de generalización menor.

Tras experimentar con distintos métodos de regularización se valida la efectividad de las técnicas de regularización en escenarios de overfitting. Las tres alternativas valoradas presentan resultados positivos, consiguiendo reducir la diferencia entre las predicciones de entrenamiento y test.

El método de Early Stopping es el más sencillo de implementar, es efectivo, pero en menor medida que el resto de técnicas. La razón es que este método simplemente detiene el entrenamiento cuando considera que se ha alcanzado el posible mínimo global. La regularización L2 y Dropout resulta más efectiva, ya que trata el problema del sobreajuste de raíz, aplicando penalizaciones en la fase de entrenamiento. Por contra la aplicación de Dropout y regularización de Ridge aumenta los tiempos de entrenamiento, mientras que el método de Early Stopping los disminuye.

Durante cada iteración, el modelo tiene que calcular dos operaciones de filtrado sobre la red. Este filtrado es una evaluación que se realiza sobre cada neurona con el fin de determinar si debe permanecer o descartarse. En el caso de L2, el incremento es de un 15 %, frente al 20 % de Dropout, por lo que se deduce que resulta menos costoso aplicar la penalización por capa (véase expresión de la página 21) y agregarla al valor de coste, con respecto al filtrado Dropout. Esta afirmación se valida con las dos configuraciones finales presentadas: el modelo de dos capas aplica Dropout y el

modelo de 3 capas L2, y este último consigue uno tiempo de ejecución 40 segundos menor, pese a ser un modelo más profundo (páginas 75 y 78).

Las regularizaciones de Dropout y L2 han resultado más complejas de ajustar, principalmente la regularización L2. Hay que ser cauteloso en el escalado de las penalizaciones, porque si se selecciona la escala incorrecta, se provoca que el modelo se enfrente a un ruido que le impida clasificar. En este caso la mejor estrategia durante la experimentación previa ha sido fijar valores arbitrarios en distintas escalas, y seguir acotando los valores en función de los valores reportados. En cuanto a su capacidad de regularización, los tres métodos ofrecen resultados similares, si bien es cierto que en el modelo de 3 capas resulta ganador L2 sobre Dropout mientras que en el modelo de 2 capas ocurre lo contrario. En términos generales se constata que las técnicas más efectivas en relación complejidad y capacidad de regularización son el método de Early Stopping y el método de regularización L2.

Se constata que la aplicación de técnicas de sobre-muestreo y reajuste de la red neuronal resultan en un clasificador capaz de identificar con solvencia las cuatro clases del problema.

Por otra parte, es necesario validar si otras arquitecturas de red como las redes neuronales recurrentes son capaces de aprovechar en mayor medida las representaciones, y dar un modelo de mayor calidad.

### 4.3. Filtrador Web de Noticias

En este apartado se describe la arquitectura del servicio web de noticias. El servicio web aporta un caso de uso sencillo extremo a extremo, en el que validar el potencial del clasificador.

#### 4.3.1. Arquitectura del sistema

*StanceDetector*<sup>7</sup> es un servicio que permite a los usuarios obtener la clasificación de una noticia en particular. Los usuarios acceden a la herramienta mediante una aplicación web de dos vistas. El servicio sigue una arquitectura de cliente-servidor.

El **extremo cliente** (Frontend) está desarrollado con tecnologías de desarrollo web como *HTML*, *CSS* y *Javascript*, haciendo uso de frameworks populares como *Bootstrap* para el diseño *CSS* y *JQuery* para controlar la lógica básica del cliente. El cliente realizará peticiones HTTPS al extremo servidor por medio de una semántica REST, para obtener la clasificación de la noticia solicitada por el usuario. La API REST es el punto de comunicación entre el extremo servidor y el extremo cliente, y abstrae a este último de las funcionalidades del primero.

El **extremo servidor** (Backend) se encuentra estratificado en varias capas. La primera es el **servidor HTTPS**, que ofrece el servicio externamente. Para exponer el servicio se requiere de dos componentes: un servidor HTTP que despliegue la API REST en ámbito local y un proxy inverso que exponga el servicio de forma segura.

---

<sup>7</sup><https://ailopera.github.io/stanceDetector.github.io/>

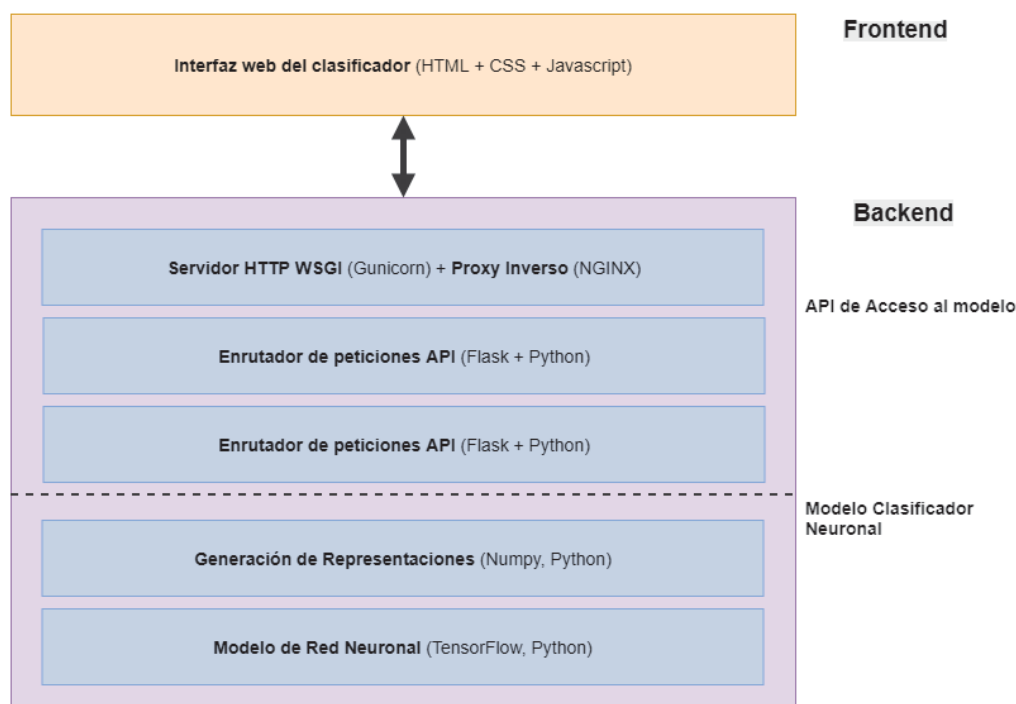


Figura 42: Pila tecnológica del clasificador web de noticias.

Para la implementación del proxy inverso se ha utilizado el servicio NGINX <sup>8</sup>, y para el despliegue del servidor Gunicorn <sup>9</sup>, que es compatible con la tecnología de desarrollo de la API.

La siguiente capa es la **API Rest** del servicio. Esta api expone una operación para obtener la posición de una noticia.

La **api** expone el método de clasificación por medio de una petición POST a la ruta */Stances*, con el titular y cuerpo de noticia que ha introducido el usuario en la plataforma. Se ha implementado un sencillo mecanismo de gestión de errores para informar al cliente de errores durante el tratamiento de la petición. Si la petición se ha satisfecho con éxito devuelve el titular, cuerpo de noticia, clasificación y un mensaje descriptivo sobre esta última.

Para obtener la clasificación asociada el módulo de la api traslada la petición al mecanismo de **middleware de acceso al modelo**. Este módulo se encarga de realizar llamadas al módulo de generación de representaciones, también implementado en Python, haciendo uso del framework *Gensim* para la manipulación de los modelos Word2vec, y la librería Numpy para gestionar las representaciones de entrada del modelo. Una vez obtenida las representaciones del texto, carga en memoria el modelo clasificador y procede a obtener la predicción.

A nivel de diseño el aspecto más complejo es cómo se gestiona el modelo clasificador entrenado y el modelo de word2vec, necesario para la obtención de la represen-

<sup>8</sup><http://nginx.org/>

<sup>9</sup><http://gunicorn.org/>



tación embebida de los textos. La gestión de modelos de TensorFlow se simplifica gracias a las funciones de guarda y recuperación de modelos que proporciona. El guardado de modelos en memoria se ha realizado durante el entrenamiento de las distintas configuraciones, haciendo uso de los nodos de guarda de TensorFlow [61]. El mecanismo de recuperación de modelos consiste en el inicio de una sesión de TensorFlow, la carga del binario y datos del modelo, y la ejecución de la predicción con las representaciones generadas.

En la configuración final se emplea el modelo pre-entrenado oficial de Word2Vec, de una capacidad de 3.5 GB, por lo que es importante gestionarlo de forma óptima en tiempo de ejecución. Además, será una estructura de acceso frecuente ya que por cada término de los textos de entrada es necesario realizar una consulta al modelo. El tiempo de carga del modelo es de alrededor de un minuto y medio, por lo que se ha optado por mantener el modelo pre-cargado en memoria durante la ejecución. Cuando se arranca el servicio se realiza la carga del modelo, y ya se encuentra disponible durante el tiempo de ejecución.

El backend se encuentra en una instancia de *Google Compute Engine*<sup>10</sup> (Modelo de infraestructura como servicio), con 4 CPUs virtuales y 15 GB de Memoria (n1-standard-4). Los núcleos son Intel Xeon, pertenecientes a la generación Intel Broadwell (2014), y operan a una frecuencia de 2.2 GHz. Los tiempos de ejecución y métricas referidas en este trabajo se han obtenido en esta misma máquina.

En cuanto al entorno de ejecución, se ha utilizado *Anaconda* para crear un espacio de trabajo aislado con las dependencias de código necesarias. Se ha utilizado la versión 1.8 de TensorFlow [59] y la versión 0.19 de Scikit-Learn [54].

---

<sup>10</sup><https://cloud.google.com/compute/docs/machine-types>



#### 4.3.2. Interfaz de acceso a la plataforma.

El servicio dispone de dos vistas que les permitirá a los usuarios interactuar de una forma intuitiva. La página principal consiste en un formulario web en el que se debe proporcionar el titular y cuerpo/sección de la noticia que se desee evaluar. El modelo se ha entrenado con un corpus de noticias y titulares en inglés, por lo que sólo se pueden mandar noticias en este idioma.

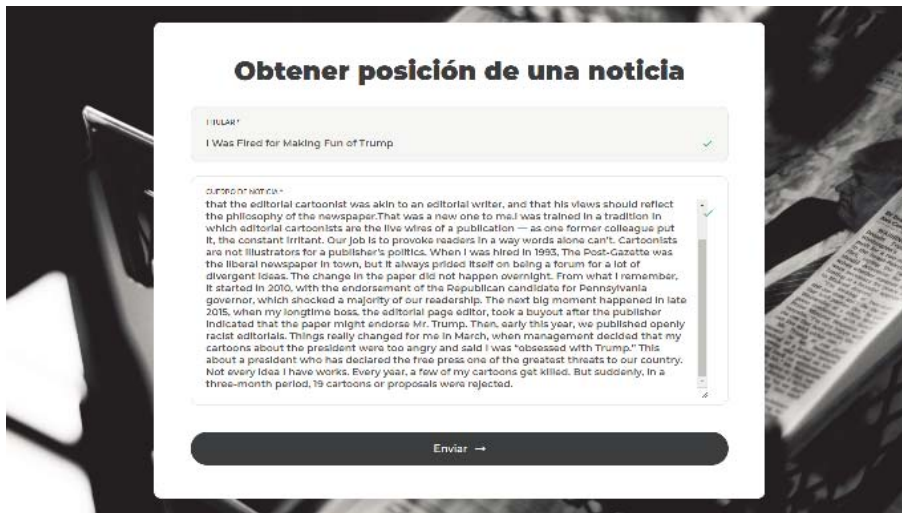


Figura 43: *Stance Detector*: Formulario de envío de noticias para su clasificación

Una vez enviada una noticia para su clasificación, la aplicación redirige al usuario a su página principal, en la que proporciona al usuario la noticia introducida para su lectura, junto con la clasificación obtenida. Pulsando en el icono de la noticia es posible acceder de nuevo al formulario de envío de noticias.



Figura 44: *Stance Detector*: captura de visualización de clasificación.

## 5. Conclusiones

En este trabajo se ha presentado un modelo clasificador neuronal utilizando representaciones basadas en vectores de embedding de word2vec. En base al modelo se ha implementado un servicio web enfocado a usuarios finales, proporcionándoles una herramienta sencilla en la que puedan ver la posición de una noticia en particular.

Durante el desarrollo del trabajo se han experimentado con varias alternativas, teniendo en cuenta enfoques clásicos como la representación de bolsa de palabras o el clasificador de Random Forest. También se ha investigado si resulta efectivo utilizar modelos de Word2Vec entrenados a partir de los datos de entrenamiento, frente al uso de modelos pre-entrenados como el modelo de Word2Vec ofrecido por Google.

Con respecto a los modelos y técnicas de representación empleadas podemos concluir lo siguiente. En relación a los modelos de clasificación constatamos la superioridad de MLP sobre Random Forest. El modelo MLP es menos dado al overfitting, porque es capaz de generalizar mejor los modelos, pero no está exento de este problema. En los resultados de la validación de modelos de representación se observa que las diversas ejecuciones de Random Forest ofrecen una diferencia de dos décimas entre los resultados de validación y entrenamiento. El modelo MLP es menos dado al overfitting, porque es capaz de generalizar mejor los modelos, pero como hemos visto, no está exento en su totalidad de este problema.

Por contra, el modelo de MLP es más sensible a datasets desbalanceados (véase matrices de confusión de las páginas 56 y 58), ofreciendo una caracterización de la clase minoritaria muy pobre.

En cuanto a los modelos de representación concluimos lo siguiente. En términos generales el modelo de BOW ofrece una caracterización uniforme de las clases, pero ofreciendo un comportamiento todo lo provechoso que podría ser. Las técnicas de sobremuestreo no ayudan en este aspecto, así que se concluye que es la propia representación la que ofrece una limitación en la caracterización de textos. El modelo de vector de medias ofrece de partida una mejor caracterización de las clases, con unos resultados potenciados aún más tras el tratamiento de SMOTE (páginas 60 y 62).

Tras los resultados obtenidos, se ratifica la eficacia en el uso de redes neuronales hacia delante para la clasificación de posicionamiento de noticias. Se obtienen unos resultados en la franja del 85 % al 87 % en término de accuracy, precisión y recall. Cabe plantearse si el error restante se debe al uso de la arquitectura de FNN como clasificador o la calidad de la representación en sí. Se considera que el problema se debe fundamentalmente por un lado a la naturaleza del dataset, que presenta un acusado desbalanceo de clases. Se ha tratado de paliar este desbalanceo aplicando aplicando SMOTE como técnica de oversampling, pero aún así la variabilidad de la clase minoritaria es baja con respecto al resto de clases. No obstante, la aplicación de técnicas de SMOTE ha supuesto un éxito y una mejora significativa del modelo.

Otra de las bases del error se debe a la representación utilizada. Si bien el modelo pre-entrenado o entrenado propio de word2vec nos permite generar una representa-

ción que refleja relaciones de contexto y semánticas, no captura todas las relaciones contextuales presentes en el cuerpo de la noticia, fundamentales para determinar la posición de esta con respecto al titular. Esto puede deberse a la estrategia de generación de representaciones, que aplica una media de los vectores de embedding, difuminando las características locales presentes en el texto.

Otro motivo puede ser que las noticias clasificadas erróneamente aporten referencias sutiles no presenten posicionamiento marcado, por lo que surge la necesidad de investigar con modelos que tengan en cuenta más características, como el medio o fecha de publicación.

## 6. Líneas Futuras

Tras los resultados concluimos que las líneas futuras de investigación deberían encauzarse hacia el perfeccionamiento de los **modelos de representación**. Para ello, se propone el uso de vectores de *embedding* como entrada de un modelo de red recurrente, que genere una representación de tamaño fijo relativa al texto y al titular. Este podría ser un buen punto de partida, para investigar si los modelos de red hacia delante consiguen aumentar su efectividad. En la literatura existente se han encontrado multitud de técnicas originales para la obtención de representaciones, si bien parece que la tendencia actual aboga por el uso de redes LSTM, redes convolucionales o una combinación de ambas. Una de las ventajas de las arquitecturas neuronales es su capacidad de reutilización, por lo que esto facilita un enfoque de experimentación iterativa, en orden creciente de complejidad.

Otra de las líneas que se propone consiste en obtener una **caracterización más amplia de las noticias**. Para ello el mayor problema es agregar las múltiples fuentes de datos para obtener una caracterización precisa y su compatibilidad con el dataset etiquetado de muestras. El dataset utilizado no ofrece información más allá de la noticia y su clasificación, por lo que sería muy complejo establecer una correlación entre las noticias y sus metadatos asociados. Habría que considerar el uso de otro dataset, con la fiabilidad que ofrece éste, apto para el agregado de información.

También se propone la extracción de características en base a entidades de interés, con el propósito de extraer **características implícitas**. Se podrían aplicar arquitecturas (Redes Neuronales Recurrentes) que implementen el concepto de factor de atención para extraer las entidades relevantes de un texto. En base a estos términos se podría consultar al modelo de Word2vec por los términos opuestos (personas, ideas o temáticas) . De esta forma se podría crear un segundo modelo cuyo propósito sería extraer el posicionamiento implícito hacia otras entidades. Este modelo partiría de las muestras de noticias que apoyan el titular, extraería las tuplas de términos opuestos y las utilizaría para alimentar un modelo que nos permitiera obtener un grado mayor de detalle en el posicionamiento de noticias.

Este trabajo ha presentado un caso de uso extremo a extremo en el ámbito de clasificación de noticias. El núcleo de dicha herramienta es el modelo y supone la parte más compleja del sistema. La estructura del servicio hace posible que éste pueda expandir su funcionalidad en un futuro. En este trabajo se ha implementado una versión inicial del servicio, enfocada exclusivamente a la clasificación directa de noticias. Para futuras versiones se propone construir alrededor de la infraestructura existente, con funciones relacionadas integración con servicios externos de noticias o servicios de gestión y consumo de éstas. Para las versiones posteriores de esta herramienta se exponen las siguientes funcionalidades:

- **Mecanismo de envío de noticias clasificadas.** Una opción a considerar a la hora de desarrollar la plataforma web es la de proveer a los usuarios de un mecanismo para enviar noticias fraudulentas o polarizadas. Esto posibilita disponer de entradas nuevas para el modelo que permitan realizar un entrenamiento iterativo del mismo.

- **Integración de fuentes externas de datos.** Servicios de guardado de artículos y fuentes RSS. Esta funcionalidad haría posible que los usuarios pudieran consultar sus fuentes de confianza desde un sólo lugar, como ya es posible desde lectores de RSS.
- **Guardado de noticias del usuario.** Consiste en implementar un espacio personal por usuario, en el que pueda guardar noticias para su posterior lectura.
- **Caracterizador de tendencias de búsqueda.** En base a los patrones de búsqueda de los usuarios en la web se podrían extraer tendencias de búsqueda sobre diversas temáticas, algo que podría utilizarse para extraer características de entrenamiento del modelo.

La caracterización de noticias es un problema desafiante que se basa en gran multitud de factores externos, por lo que es un problema que aún permite una explotación aún mayor. La posibilidad de detectar una noticia como verdadera o falsa es un auténtico desafío, y sigue siendo una cuestión saber de qué manera las técnicas de Deep Learning conseguirán conquistar la verdad.

## Referencias

- [1] S. J. A, K. Bartosz, and M. Wozniak. Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets. *Pattern Recognition*, 57:164–178, 2016.
- [2] I. Augenstein, T. Rocktäschel, A. Vlachos, and K. Bontcheva. Stance detection with bidirectional conditional encoding. *arXiv preprint arXiv:1606.05464*, 2016.
- [3] I. Augenstein, A. Vlachos, and K. Bontcheva. Usfd at semeval-2016 task 6: Any-target stance detection on twitter with autoencoders. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 389–393, 2016.
- [4] A. V. M. Barone. Low-rank passthrough neural networks. *arXiv preprint arXiv:1603.03116*, 2016.
- [5] T. Berners-Lee. Tim berners-lee: I invented the web. here are three things we need to change to save it, 2017.
- [6] P. Bourgonje, J. M. Schneider, and G. Rehm. From clickbait to fake news detection: an approach based on detecting the stance of headlines to articles. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 84–89, 2017.
- [7] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
- [8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
- [9] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [10] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Machine Learning Proceedings 1995*, pages 115 – 123. Morgan Kaufmann, San Francisco (CA), 1995.
- [11] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS’12*, pages 1223–1231, USA, 2012. Curran Associates Inc.
- [12] E. DeRouin, J. Brown, H. Beck, L. Fausett, and M. Schneider. Neural network training on unequally represented classes. *Intelligent engineering systems through artificial neural networks*, pages 135–145, 1991.
- [13] T. Developers. Tensorflow xla, 2018.

- [14] J. Du, R. Xu, Y. He, and L. Gui. Stance classification with target-specific neural attention networks. *International Joint Conferences on Artificial Intelligence*, 2017.
- [15] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.
- [16] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [17] W. Ferreira and A. Vlachos. Emergent: a novel data-set for stance classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 1163–1168, 2016.
- [18] M. Ghiassi, M. Olschimke, B. Moon, and P. Arnaudo. Automated text classification using a dynamic artificial neural network model. *Expert Systems with Applications*, 39(12):10967 – 10976, 2012.
- [19] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures<sup>1</sup>.
- [20] A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, Inc., 1st edition, 2017.
- [21] Z. S. Harris. Distributional structure. *WORD*, 1954.
- [22] K. S. Hasan and V. Ng. Stance classification of ideological debates: Data, models, features, and constraints. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1348–1356, 2013.
- [23] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.
- [24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [25] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, Mar. 1991.
- [26] M. I. Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.
- [27] Y. Kim. Convolutional neural networks for sentence classification. *arXiv pre-print arXiv:1408.5882*, 2014.



- [28] G. Krishnalal, S. B. Rengarajan, and K. Srinivasagan. A new text mining approach based on hmm-svm for web news classification. *International Journal of Computer Applications*, 1(19):98–104, 2010.
- [29] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- [30] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273, 2015.
- [31] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.
- [32] G. Lemaitre, F. Nogueira, D. Oliveira, and C. Aridas. Imbalanced learn api: Smote, 2016.
- [33] P. Liu, S. Joty, and H. Meng. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1433–1443, 2015.
- [34] P. Liu, X. Qiu, and X. Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.
- [35] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [36] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.
- [37] S. Mohammad, S. Kiritchenko, P. Sobhani, X. Zhu, and C. Cherry. Semeval-2016 task 6: Detecting stance in tweets. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 31–41, 2016.
- [38] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence  $o(1/k^2)$ . In *Doklady AN USSR*, volume 269, pages 543–547, 1983.
- [39] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung. A supervised learning approach for imbalanced data sets. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.
- [40] S. Nitish, H. Geoffrey, K. Alex, S. Ilya, and S. Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.



- [41] NLTK Project. Nltk tokenizer package, 2017.
- [42] NLTK Project Developers. Nltk project, 2017.
- [43] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.
- [44] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- [45] H. Pouransari and S. Ghili. Deep learning for sentiment analysis of movie reviews. Technical report, Technical report, Stanford University, 2014.
- [46] J. R. QUINLAN. Chapter 1 - introduction. In J. R. QUINLAN, editor, *C4.5*, pages 1 – 16. Morgan Kaufmann, San Francisco (CA), 1993.
- [47] N. Rakholia and S. Bhargava. 'is it true?—deep learning for stance detection in news', 2016.
- [48] G. Rao, W. Huang, Z. Feng, and Q. Cong. Lstm with sentence representations for document-level sentiment classification. *Neurocomputing*, 2018.
- [49] R. Rehurek. Gensim official site, 2018.
- [50] R. Rehurek. Gensim official site, 2018.
- [51] L. Richardson. Beautiful soup documentation, 2015.
- [52] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, Nov. 1997.
- [53] Scikit-Learn developers. Scikit learn official api reference, 2017.
- [54] Scikit Learn Developers. Scikit learn official site, 2018.
- [55] Skymind. Word2vec: Neural word embeddings in java and scala., 2017.
- [56] P. Sobhani, S. Mohammad, and S. Kiritchenko. Detecting stance in tweets and analyzing its interaction with sentiment. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*, pages 159–169, 2016.
- [57] Spark Developers. Feature extraction and transformation - rdd-based api., 2018.
- [58] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.
- [59] TensorFlow Developers. Tensorflow official site, 2018.

- [60] TensorFlow Developers. Tensorflow official site: Getting started for ml beginners, 2018.
- [61] TensorFlow Development Team. Tensorflow v1.8 official api documentation (python), 2018.
- [62] TensorFlow Development Team. Tensorflow v1.8 official programmers guide, 2018.
- [63] The Apache Software Foundation. Spark official site, 2018.
- [64] J. Thorne, M. Chen, G. Myriantous, J. Pu, X. Wang, and A. Vlachos. Fake news stance detection using stacked ensemble of classifiers. In *Proceedings of the 2017 EMNLP Workshop: Natural Language Processing meets Journalism*, pages 80–83, 2017.
- [65] G. Varoquaux. Repositorio oficial de joblib, 2017.
- [66] F. N. C. Volunteers. Fake news challenge: Exploring how artificial intelligence technologies could be leveraged to combat fake news., 2017.
- [67] F. N. C. Volunteers. fnc-1-baseline: A baseline implementation for fnc-1, 2017.
- [68] W. Wei, X. Zhang, X. Liu, W. Chen, and T. Wang. pkudblab at semeval-2016 task 6: A specific convolutional neural network system for effective stance detection. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 384–388, 2016.
- [69] G. Zarrella and A. Marsh. Mitre at semeval-2016 task 6: Transfer learning for stance detection. *arXiv preprint arXiv:1606.03784*, 2016.
- [70] Q. Zeng, Q. Zhou, and S. Xu. Neural stance detectors for fake news challenge.
- [71] C. Zhou, C. Sun, Z. Liu, and F. Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

Este documento esta firmado por



<b>Firmante</b>	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
<b>Fecha/Hora</b>	Wed Jul 04 23:38:55 CEST 2018
<b>Emisor del Certificado</b>	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
<b>Numero de Serie</b>	630
<b>Metodo</b>	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)