

# **TOXICOM: DETECCIÓN DE MENSAJES TÓXICOS EN MEDIOS SOCIALES**

Autor: Raúl Torné Alonso

Tutora: Ana Valdivia García

Profesor: Jordi Casas Roma

Nombre del máster: Máster de Ciencia de Datos (Data Science)

Área: Informática

Github: <https://github.com/racu10/toxicom>

03/06/2018

# Copyright

## The FreeBSD Documentation License

Copyright (c) 2018, Raúl Torné Alonso

All rights reserved.

Redistribution and use in source (SGML DocBook) and 'compiled' forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (SGML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY THE FREEBSD DOCUMENTATION PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FREEBSD DOCUMENTATION PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# FICHA DEL TRABAJO FINAL

<b>Título del trabajo:</b>	<i>TOXICOM: DETECCIÓN DE MENSAJES TÓXICOS EN MEDIOS SOCIALES</i>
<b>Nombre del autor:</b>	<i>Raúl Torné Alonso</i>
<b>Nombre del colaborador/a docente:</b>	Ana Valdivia García
<b>Nombre del PRA:</b>	Jordi Casas Roma
<b>Fecha de entrega (mm/aaaa):</b>	<i>06/2018</i>
<b>Titulación o programa:</b>	Máster de Ciencia de Datos (Data Science)
<b>Área del Trabajo Final:</b>	<i>Trabajo final de máster</i>
<b>Idioma del trabajo:</b>	<i>Español</i>
<b>Palabras clave</b>	<i>Artificial Intelligence, Natural Language Processing, Toxic Comments</i>

## **Resumen del Trabajo:**

El presente proyecto tiene como finalidad, detectar en los medios sociales de manera automática aquellos mensajes cuya finalidad es negativa. Para ello, se utilizarán técnicas de Inteligencia Artificial basados en algoritmos de Machine Learning y Deep Learning.

Cabe tener en cuenta que, actualmente existe una gran diversificación de medios sociales, en los cuales gran parte de la población se mantiene continuamente activa. Es así, que algunos de estos tan importantes como Twitter, Facebook e Instagram de entre muchos otros, generan millones de publicaciones diariamente. Así pues, resulta necesario que aquellos medios sociales en los que los usuarios generan contenido, sean controlados para de este modo, evitar mensajes con intencionalidad negativa y que al mismo tiempo generan odio.

No obstante, procesar la detección manual de estos mensajes conllevaría una gran cantidad de tiempo. Por ello, actualmente la mayoría de plataformas integran una solicitud de baneo manual. Aunque cabe decir que, únicamente se evalúan aquellos que han sido baneados, cuestión que genera un gran retraso y motivo por el que muchos usuarios ya pueden haberse visto perjudicados.

Así pues, para evitar la gran cantidad de tiempo de análisis y procesado de cada uno de los mensajes generados, es necesario implementar nuevos modelos que permitan clasificar los mensajes según su contenido automáticamente. De este modo, permitirá al gestor del medio

social, por un lado, indicar que tipos de mensaje serán aceptados y por otro cuales no serán publicados.

**Abstract:**

The purpose of this project is to automatically detect those messages whose purpose is negative in the social media. For this, Artificial Intelligence techniques based on Machine Learning and Deep Learning algorithms will be used.

It should be borne in mind that there is currently a great diversification of social media, in which a large part of the population is continuously active. Thus, some of these important as Twitter, Facebook and Instagram among many others, generate millions of publications daily. So, it is necessary that those social media in which users generate content, will be needed controlled in this way, for avoid messages with negative intent and that at the same time generate hate.

However, processing the manual detection of these messages would take a large amount of time. For this reason, most platforms currently integrate a request for manual prohibition. Although it must be said that only those that have been banned are evaluated, a problem that generates a great delay and why many users may have suffered damages when the ban is applied.

At this time, to avoid the large amount of analysis and processing time of each of the generated messages, it is necessary to implement new models that allow the messages to be classified according to their content automatically. In this way, it will allow the manager of the social environment to indicate which types of messages will be accepted or not be published.

# Agradecimientos

En primer lugar, me gustaría decir que en este proyecto muchas personas me han mostrado su apoyo o si mas no su ayuda, por lo tanto, me gustaría mostrar mi más sincero agradecimiento.

Por un lado, a mi familia y pareja ya que me han apoyado y animado en los momentos difíciles, además también me gustaría agradecer que me hayan ayudado a permitirme realizar mis estudios.

# Abstract

The purpose of this project is to automatically detect those messages whose purpose is negative in the social media. For this, Artificial Intelligence techniques based on Machine Learning and Deep Learning algorithms will be used. It should be borne in mind that there is currently a great diversification of social media, in which a large part of the population is continuously active. Thus, some of these important as Twitter, Facebook and Instagram among many others, generate millions of publications daily. So, it is necessary that those social media in which users generate content, will be needed controlled in this way, for avoid messages with negative intent and that at the same time generate hate.

However, processing the manual detection of these messages would take a large amount of time. For this reason, most platforms currently integrate a request for manual prohibition. Although it must be said that only those that have been banned are evaluated, a problem that generates a great delay and why many users may have suffered damages when the ban is applied.

At this time, to avoid the large amount of analysis and processing time of each of the generated messages, it is necessary to implement new models that allow the messages to be classified according to their content automatically. In this way, it will allow the manager of the social environment to indicate which types of messages will be accepted or not be published.

# Resumen

El presente proyecto tiene como finalidad, detectar en los medios sociales de manera automática aquellos mensajes cuya finalidad es negativa. Para ello, se utilizarán técnicas de Inteligencia Artificial basados en algoritmos de Machine Learning y Deep Learning.

Cabe tener en cuenta que, actualmente existe una gran diversificación de medios sociales, en los cuales gran parte de la población se mantiene continuamente activa. Es así, que algunos de estos tan importantes como Twitter, Facebook e Instagram de entre muchos otros, generan millones de publicaciones diariamente. Así pues, resulta necesario que aquellos medios sociales en los que los usuarios generan contenido, sean controlados para de este modo, evitar mensajes con intencionalidad negativa y que al mismo tiempo generan odio.

No obstante, procesar la detección manual de estos mensajes conllevaría una gran cantidad de tiempo. Por ello, actualmente la mayoría de plataformas integran una solicitud de baneo manual. Aunque cabe decir que, únicamente se evalúan aquellos que han sido baneados, cuestión que genera un gran retraso y motivo por el que muchos usuarios ya pueden haberse visto perjudicados.

Así pues, para evitar la gran cantidad de tiempo de análisis y procesado de cada uno de los mensajes generados, es necesario implementar nuevos modelos que permitan clasificar los mensajes según su contenido automáticamente. De este modo, permitirá al gestor del medio social, por un lado, indicar que tipos de mensaje serán aceptados y por otro cuales no serán publicados.

# Índice

1.	Introducción .....	11
1.1.	Antecedentes.....	11
1.2.	Motivación .....	15
1.3.	Objetivos.....	16
1.3.1.	Hipótesis u objetivo principal .....	16
1.3.2.	Objetivos parciales .....	16
1.4.	Estado del arte .....	17
1.5.	Estructura del proyecto.....	20
2.	El odio en los medios sociales.....	21
3.	Planificación de la investigación .....	23
4.	Marco teórico.....	25
4.1.	Estructura de los textos .....	25
4.1.1.	Matriz término documento .....	25
4.1.2.	TF-IDF vector.....	27
4.1.3.	Tokenización de las palabras .....	29
4.1.4.	Word2vec.....	31
4.2.	Machine Learning .....	32
4.2.1.	Tipos de aprendizaje .....	32
4.2.2.	Problemas de clasificación .....	32
4.2.3.	Clasificadores .....	36
4.2.4.	Optimizadores.....	41
4.3.	Deep Learning .....	42
4.3.1.	Clasificadores .....	46
5.	Desarrollo practico.....	53
5.1.	Infraestructura .....	53
5.1.1.	Lenguaje .....	53
5.1.2.	Control de versiones.....	54
5.1.3.	Estructura de Archivos.....	56
5.1.4.	Licencia del proyecto .....	58
5.1.5.	Visualización y creación del código .....	59
5.2.	Desarrollo de experimentos.....	60



5.2.1.	Limpieza de los datos .....	60
5.2.2.	Visualización de los datos de entrenamiento.....	63
5.2.3.	Clasificación por Machine Learning y Deep Learning .....	69
5.2.4.	Ensembling .....	74
5.2.5.	Parámetros utilizados en el desarrollo de los experimentos .....	75
6.	Resultados.....	76
7.	Conclusiones .....	79
	Bibliografía .....	80
	Anexos .....	83

# Tabla de figuras

Figura 1 Medios sociales.....	11
Figura 2 Estadísticas de los comentarios por usuarios .....	12
Figura 3 Histograma del total de número de usuarios en cada nivel de mensajes de odio .....	12
Figura 4 Histograma del porcentaje total de ataques por usuario según el nivel de mensajes de odio .....	13
Figura 5 Histograma del total de comentarios por actividad del usuario .....	13
Figura 6 Histograma del total de comentarios de odio por actividad del usuario .....	13
Figura 7 Toxic Comment Classification Challenge .....	14
Figura 8 Diagrama de Gantt planificación ideal .....	23
Figura 9 Diagrama de Gantt planificación realizada .....	24
Figura 10 Representación Word2vec .....	31
Figura 11 Función Sigmoide.....	36
Figura 12 SVM hiperplano optimo .....	38
Figura 13 Red neuronal.....	42
Figura 14 Función Sigmoidal.....	43
Figura 15 Función ReLU .....	44
Figura 16 Binary threshold .....	44
Figura 17 Convolución de una matriz.....	46
Figura 18 Aplicación matriz pooling.....	47
Figura 19 Capa de convolución.....	48
Figura 20 CNN reducida.....	48
Figura 21 RNN .....	50
Figura 22 LSTMs.....	51
Figura 23 Neurona LSTM.....	51
Figura 24 Lenguajes de programación.....	53
Figura 25 Publicaciones github .....	54
Figura 26 Contribuciones al proyecto .....	55
Figura 27 Cookicutter organización del proyecto .....	56
Figura 28 Sistema de archivos final .....	57
Figura 29 Licencia del proyecto.....	58
Figura 30 Jupyter Notebook <a href="https://github.com/racu10/toxicom/tree/master/notebooks">https://github.com/racu10/toxicom/tree/master/notebooks</a> .....	59
Figura 31 Datos de entrenamiento .....	60
Figura 32 Datos de test .....	61
Figura 33 Datos de entrenamiento con las nuevas columnas añadidas .....	62
Figura 34 Datos de test con las nuevas columnas añadidas.....	62
Figura 35 % Of data toxicity .....	63
Figura 36 N.º de comentarios por tipo.....	64
Figura 37 Word Cloud Toxic.....	65
Figura 38 Word Cloud Severe Toxic .....	65

Figura 39 Word Cloud Obscene .....	65
Figura 40 Word Cloud Threat.....	65
Figura 41 Word Cloud Insult.....	65
Figura 42 Word Cloud Identity Hate .....	65
Figura 43 Palabras más repetidas Toxic .....	66
Figura 44 Palabras más repetidas Severe Toxic.....	66
Figura 45 Palabras más repetidas Obscene .....	67
Figura 46 Palabras más repetidas Threat .....	67
Figura 47 Palabras más repetidas Insult .....	68
Figura 48 Palabras más repetidas Identity Hate.....	68
Figura 49 Algoritmos Machine Learning sobre CV de Train basado 50% toxic 50% clean .....	69
Figura 50 Algoritmos ML sobre test basado 50% toxic 50% clean.....	70
Figura 51 Algoritmos DL sobre CV de Train basado 50% toxic 50% clean.....	70
Figura 52 Algoritmos Deep Learning sobre test basado 50% toxic 50% clean .....	71
Figura 53 Comparación ML estructuras TF-IDF y Matriz TD con 158627 características .....	72
Figura 54 Deep Learning con Tokenización de palabras por secuencia .....	72
Figura 55 Regresión Logística - NGramas .....	73
Figura 56 Ensembling de los diferentes algoritmos .....	74

## Tablas de datos

Tabla 1 Cross Validation de mejores características 50% toxic 50% clean .....	76
Tabla 2 Predicción sobre test de mejores características 50% toxic 50% clean .....	76
Tabla 3 Comparación entre TF-IDF y Matriz TD con 158627 características sobre test.....	77
Tabla 4 Algoritmos Deep Learning predicciones sobre Test .....	77
Tabla 5 Logistic Regression modificando los N-gramas utilizados y el N.º Características sobre test .....	78
Tabla 6 Ensembling sobre las predicciones de Test Top de cada algoritmo.....	78

# 1.Introducción

A continuación, se presentan las principales ideas que han incitado dicha investigación. En primer lugar, se mostrará la contextualización, haciendo referencia a los antecedentes en el apartado 1.1 y a la motivación de la realización a lo largo del apartado 1.2. En segundo lugar, en el apartado 1.3 se presentarán los objetivos principales y parciales. Posteriormente, a partir de diferentes autores en el apartado 1.4, se mostrará el estado del arte actual. Y finalmente, en el apartado 1.5 se describirá la organización del proyecto.

## 1.1. Antecedentes

Con el paso de los años internet ha mostrado una gran evolución. Como consecuencia, esto ha permitido la creación de la web, cuyos inicios comenzaron con la versión 1.0, también conocida como World Wide Web. Esta es una web estática en la que cualquier usuario puede consultar información. Su principal uso es publicar documentos e información sobre todo cultural. Posteriormente se generó la web 1.X, la cual proporciona herramientas a los administradores de los portales para que así puedan realizar publicaciones de nuevos contenidos, sin la necesidad de tener conocimientos de programación. Actualmente, nos encontramos en la web 2.0, también denominada web interactiva, dado que los propios usuarios pueden interactuar con esta sin la necesidad de tener conocimientos de programación. Además de ello, permite la creación, compartición, modificación y eliminación de contenidos. Dando como resultado el nacimiento de todos los medios sociales.

En este tipo de web, el cambio más significativo es que los usuarios pasan de ser espectadores a protagonistas sin la necesidad de utilizar un perfil real, la cual cosa permite mantener la anonimidad.

Ahora bien, el hecho de que los usuarios tomen el rol de protagonistas, conlleva a la creación de contenidos de manera positiva. Esto permite que puedan interactuar para generar un mejor contenido. No obstante, no todos los usuarios interactúan de manera positiva. Estos generan mensajes de odio en los distintos medios sociales, con el fin de atacar directamente a individuos o grupos de personas.

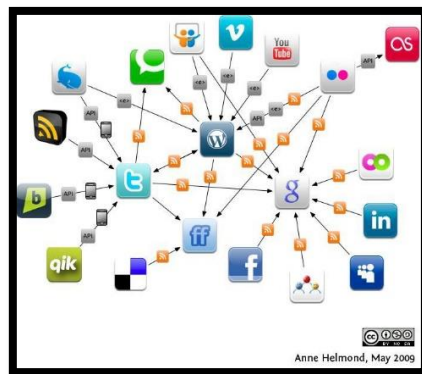


Figura 1 Medios sociales

A modo de ejemplo, un estudio y proyecto realizado por Google en 2015 denominado Jigsaw, tenía como finalidad encontrar una solución contra los de ataques que generan odio. Esto se desarrolló concretamente sobre la plataforma de Wikipedia, de manera que, se realizó la clasificación de cada uno de los comentarios extraídos y un posterior análisis de estos. Así pues, tal y como muestran las estadísticas realizadas en la Figura 2, se concluyó que la mayor parte de los ataques provienen de usuarios anónimos, aunque también se detectó a algunos usuarios registrados.

Anonymity	Number of Accounts	Number of Comments	Attack Prevalence
Anonymous	97,742	191,460	3.1%
Registered	129,394	2,023,559	0.5%
Totals	227,136	2,215,019	0.8%

Figura 2 Estadísticas de los comentarios por usuarios

Entrando más en detalle, se clasificaron los mensajes según su nivel de odio, el cual representaba la gravedad del comentario. Una vez realizado el estudio, se pudo detectar que los mensajes de mayor odio provenían de 34 usuarios, que además eran extremadamente activos. Este hecho, se puede apreciar en la Figura 3, lo cual decir, tal y como se muestra en la Figura 4, aproximadamente un 10% de todos los ataques personales, con un nivel de odio de más de 20 que provenía de estos usuarios.

Por otro lado, se puede observar en la Figura 4 que el 80% de los ataques por persona corresponden a 5 comentarios o menos, los cuales provienen de aproximadamente 9000 usuarios de la plataforma. Estos datos, se pueden comprobar tanto en la Figura 4 como en la Figura 5 y Figura 6.

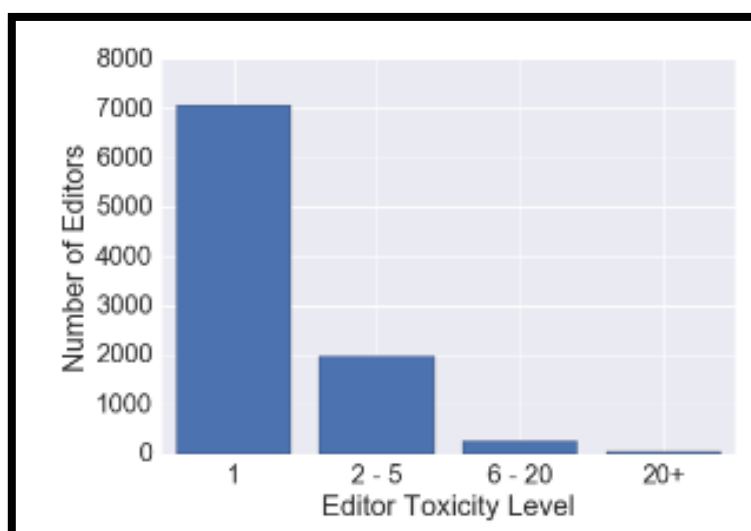


Figura 3 Histograma del total de número de usuarios en cada nivel de mensajes de odio

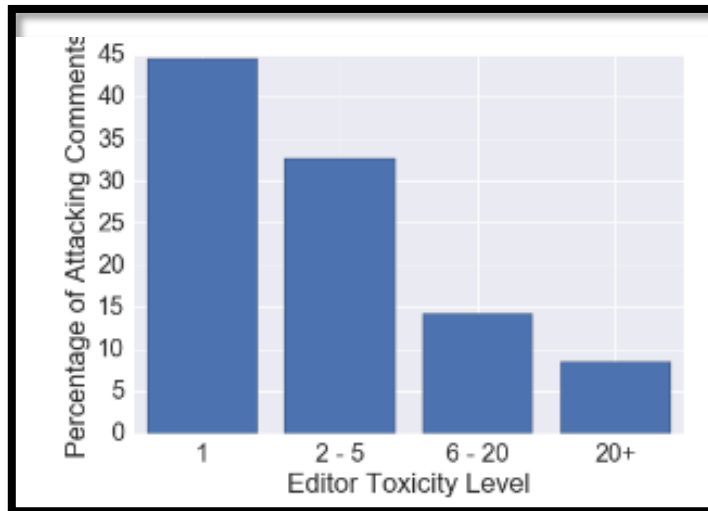


Figura 4 Histograma del porcentaje total de ataques por usuario según el nivel de mensajes de odio

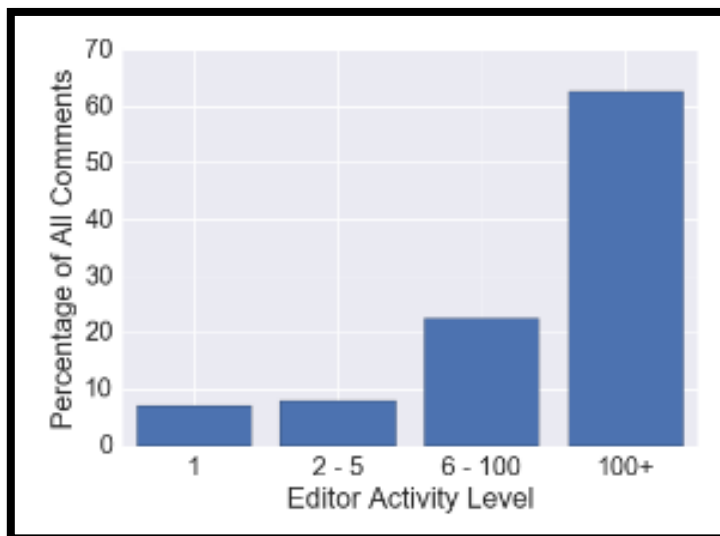


Figura 5 Histograma del total de comentarios por actividad del usuario

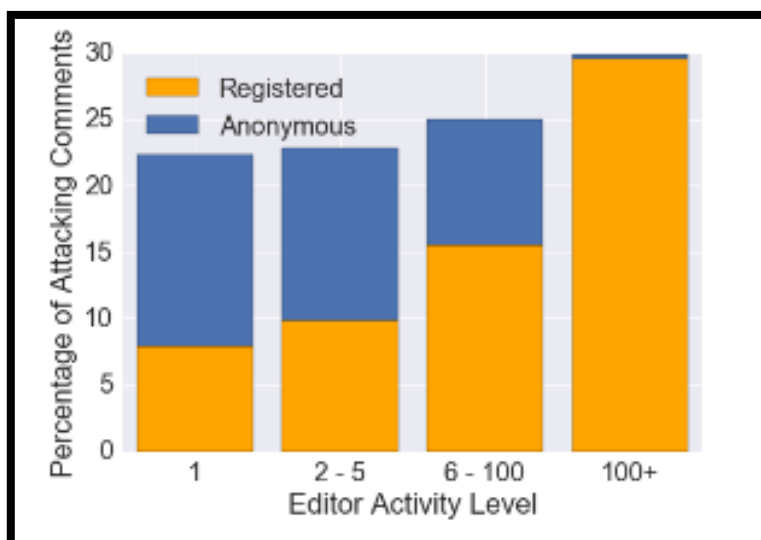


Figura 6 Histograma del total de comentarios de odio por actividad del usuario

Retomando el proyecto Jigsaw, Google propuso el 19 de diciembre del 2017 una competición en la plataforma de Kaggle, con los datos extraídos de la Wikipedia que habían sido clasificados previamente. El objetivo principal, es demostrar que la Inteligencia Artificial permite hallar odio en el contenido de los mensajes, mediante los comentarios previamente clasificados. Además de ser capaz de reconocer un mensaje no clasificado y tener la autonomía de discernir entre un mensaje de odio permite indicar los tipos a los cuales va referido, según si son obscenos, con insultos, odio de la persona, etc.

Como se puede observar, la Figura 7 muestra datos más específicos sobre la competición. Por ejemplo, expone el premio el cual era la suma 35000\$ que iba dirigido a los tres primeros en la clasificación. De este modo, lograron incentivar a diferentes desarrolladores para que participasen.

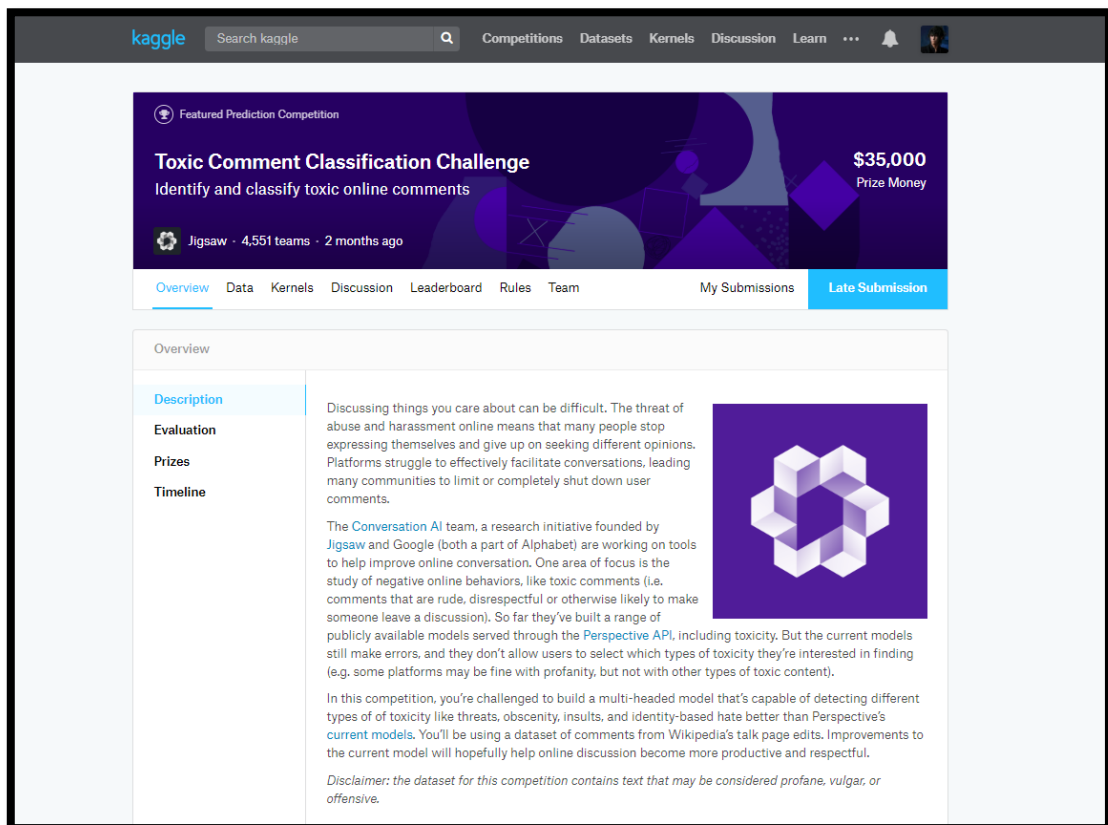


Figura 7 Toxic Comment Classification Challenge

## 1.2. Motivación

En primer lugar, me gustaría mencionar que la principal motivación que he tenido para la creación y desarrollo de este proyecto ha sido la posibilidad de aprender a tratar el lenguaje natural de las personas. Pues como ya se ha comentado previamente, gestionar la alta cantidad de mensajes que se crean por minuto es tan alta que analizarlos todos llega a ser una tarea bastante compleja, dada la abundancia generada. De manera que, quería llegar a tratar este reto, para dar con unas soluciones que sean lo suficientemente óptimas. Por ello a partir de una investigación previa de todos los algoritmos utilizados me dispuse a probar y modificar para poder llegar a dar una conclusión sobre la clasificación de mensajes tóxicos.

Otra de mis motivaciones era el poder participar dentro de la competición de Google en Kaggle, aunque por tema de tiempo la competición finalizase antes de poder empezar a implementar algoritmos.

Así mismo, otro motivo era ponerme el reto de con algún algoritmo implementado obtener una alta precisión, la cual fuese lo suficientemente alta para obtener una puntuación la cual estuviese por encima de la media de los usuarios.



## **1.3. Objetivos**

En este apartado, se expondrán cada uno de los objetivos principales y parciales de la investigación a realizar.

### **1.3.1. Hipótesis u objetivo principal**

La identificación de mensajes abusivos de forma automática permitirá prevenir los abusos en los medios sociales, a la vez que si estos funcionan en tiempo real se puede llegar a prevenir que otro usuario lea este mensaje abusivo y le afecte personalmente. Por ello, es relevante crear un modelo de clasificación que sea capaz de filtrar textos que sean abusivos.

### **1.3.2. Objetivos parciales**

El primer objetivo, es proceder a realizar un análisis de la base de datos. En este caso corresponderá a cuántos mensajes abusivos existen en la plataforma de Wikipedia, e intentar entender mediante visualizaciones, como están distribuidos los datos.

El segundo objetivo, es estudiar las técnicas más conocidas de Natural Language Processing (NLP). Para realizar una transformación de todos los comentarios a estructuras.

El tercer objetivo, es aplicar técnicas de Machine Learning y Deep Learning. Para realizar una comparación entre los resultados obtenidos de ambos.

Finalmente, mediante los algoritmos utilizados se estudiarán la performance de los algoritmos propuesto en la competición de Kaggle. Para intentar realizar mejoras en su precisión de clasificación.

## 1.4. Estado del arte

Durante los últimos años, debido a la gran cantidad de mensajes volcados en la red, se han creado técnicas para analizar y procesar esa información. Es así como diferentes autores han desarrollado y utilizado algoritmos de Procesamiento del Lenguaje Natural (NLP), tal y como se relata en el libro de (Goldberg, Y. 2016). En el apartado de “trabajando con el lenguaje natural”, nos expresa que, según el problema, se debe de afrontar de formas totalmente distintas. El destaca abordarlo mediante el modelado del lenguaje, el cual permite adaptarlo a una estructura siguiendo un algoritmo. Es decir, o bien mediante la representación de palabras en vectores entrenados previamente o haciendo uso de diferentes estructuras vectoriales llamadas Word Embeddings. Posteriormente tras realizar la transformación del lenguaje a una estructura, se debe procesar a través de algoritmos de Inteligencia Artificial, los cuales están divididos en dos grandes grupos. En primer lugar, los algoritmos de Machine Learning y en segundo lugar los algoritmos de Deep Learning.

Los algoritmos de Machine Learning para la detección de mensajes ofensivos (Shalev-Shwartz, S., y Ben-David, S. 2014), son Native Bayes (NB), Support Vector Machine (SVM), Linear Regresion (LR), de entre otros muchos. La ventaja de este tipo de algoritmos es que suele ser rápido el aprendizaje. Las estructuras más utilizadas en Machine Learning son las basadas en el algoritmo de Bag Of Words (Kumara, y otros, 2017), como es el TF-IDF.

Por otro lado, encontramos quienes afrontan la detección de mensajes ofensivos mediante Deep Learning. A partir del uso de algoritmos, los cuales deben de modificar y adaptar los mensajes en estructuras vectoriales. El artículo de (Young, T., y otros 2018) lo detalla claramente explicando las estructuras de vectores de distribución y Word2vect aplicados a una Convolutional Neural Networks (CNN). Así pues, se encuentran una gran variedad de algoritmos de Deep Learning, los cuales se basan en la creación de redes neuronales que intentan simular las neuronas del cerebro humano, tal y como se detalla en el libro de (Goodfellow, I, y otros, 2016). Dichas redes, se pueden diseñar de distintas formas, todo dependerá de la necesidad y del aprendizaje que se desee dar a la red. Siendo diseñadas a partir de capas y perceptrones. Algunos de los diseños más utilizados son las Convolutional Neural Networks (CNN), las Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), etc.

A continuación, se explicarán una serie de artículos que tratan directamente sobre la clasificación de mensajes ofensivos de entre otros muchos.

El artículo (Founta, A. M., y otros, 2018) tiene como objetivo encontrar mensajes abusivos o comportamientos tóxicos en las redes. Todo ello, haciendo uso de mensajes extraídos de Twitter, con la finalidad de intentar demostrar que no tan solo es necesario utilizar el texto para poder detectar estas actitudes de forma genérica, sino que también es necesario añadir otras características como por ejemplo en esta plataforma el número de seguidores, la localización, la edad, etc. Además, intentando clasificar en Cyberbullying, Ofensivos, Odio y Sarcasmo. Como algoritmo de Machine Learning utiliza Naive Bayes, el cual la media de clasificación por tipología es aproximadamente de un 0.86. Por otro lado, como algoritmo de Deep Learning utiliza

Convolutional Neural Network (CNN), solo utilizando el texto tiene una media aproximada de 0.89 y en cambio añadiéndole más información externa a la red neuronal tiene una media aproximada de 0.92. De manera que, esto genera una gran diferencia de precisión.

El artículo (Georgakopoulos, S. V., y otros, 2018), tiene como objetivo demostrar que el algoritmo *Convolutional Neural Networks* (CNN) trata mejor este tipo de mensajes ofensivos que los algoritmos basados en Machine Learning, haciendo uso de los comentarios de Wikipedia. Así pues, el algoritmo con una mayor precisión de Machine Learning es el Support Vector Machine, con una precisión de 0.81. En cambio, si se hace uso de una Convolutional Neural Network (CNN), se obtiene una precisión de 0.91.

El artículo (Ratkiewicz, J., y otros, 2011) el objetivo principal es realizar un análisis en la red de Twitter, obteniendo en tiempo real noticias de US o intereses generales sobre este, además de clasificar los memes obtenidos entre verdadero, legítimo o eliminar, haciendo uso de los algoritmos *Adaboost* y *Support Vector Machine* (SVM). En este caso se puede apreciar que para el tratamiento de clasificaciones a tiempo real. El remuestreo de los datos tiene un factor importante y es por ello que Adaboost obtiene una precisión de 0.96 y Support Vector Machine (SVM) 0.95.

El artículo (Wulczyn, E., Thain, N., y Dixon, L. 2017) tiene como objetivo clasificar todos los comentarios de la Wikipedia en inglés en las diferentes categorías que representan mensajes tóxicos, los cuales previamente han sido clasificados por humanos mediante crowdsourcing. Donde a cada comentario se le han atribuido ciertas preguntas que deberán ser respondidas para identificar si son tóxicas o no. Así pues, la clasificación final viene definida por *One Hot* (OH) especificando que la clasificación es idéntica a la clasificación de la persona o *Empirical distribution* (ED) que indica la diferente opinión sobre si un comentario es un ataque. Para poder realizar el estudio se han comparado los algoritmos de *linear regresion* (LR) y *Multi-Layer Perceptron* (MLP). En este caso como referencia de precisión utiliza area under the curve (AUC). En este caso la linear regresión tiene un AUC de 96.24 y el Multi-Layer Perceptron (MLP) 96.59, aunque son muy parecidos el MLP es un poco mejor.

El artículo (Chatzakou, D., y otros, 2017) trata sobre la detección de bullying y abusos en la red de Twitter, mediante un aprendizaje no supervisado haciendo uso como estructura word2vec. Los clasificadores utilizados son basados en Arboles (J48, LADTree, LMT, NBTree, Random Forest (RF), and Functional Tree). El objetivo era comparar si predeciendo menos clases tenía mejor resultado. Así pues, como tres clases se utilizaron Bulling, Agresivos y normales. En cambio, con cuatro clases se añadió la de Spam. La comparación mediante AUC es que con 3 clases los clasificadores basados en arboles tenían una media de 90.7% en cambio con 4 decrecía a 81.5%.

El artículo (Hariani, y Riadi, I. 2017) se basa en la detección de cyberbullying en la red de Twitter. Utilizando como estructura de palabras TF-IDF y mediante una clasificación de NB. La cual obtiene una precisión del 96.55%.

El artículo (Davidson, T., Warmesley, D., Macy, M., y Weber, I. 2017) trata de clasificar los datos de Twitter en Odio, Ofensivos y normales. Para ello utiliza como estructura de palabras TF-IDF y como clasificador una regresión logística. El cual consigue tener unos resultados en la categoría de odio de 61%, en ofensivos un 91% y en normales un 95%.

El artículo (Mehdad, Y., y Tetreault, J. R. 2016) tiene como objetivo comparar los algoritmos NB, SVC, RNN, utilizando como estructura de palabras la tokenización y los modelos pre entrenados. Otro factor que tiene en cuenta es si la unión mediante ensembling da mejores resultados. Para este caso la mejor puntuación la obtiene haciendo ensembling de NBSVM con un AUC de 91%. En cambio, con RNN el máximo conseguido es del 85%. Mediante un ensembling de todos los algoritmos implementados es capaz de superar el AUC obtenido por NBSVM, con un valor del 93%.

Se puede observar que en la mayoría de casos a la hora de clasificar comentarios los algoritmos de Deep Learning dan un poco de mejor resultado que los de Machine Learning, aunque todo varía según el artículo y la forma de implementarlo.

Para concluir, cabe decir que este tema que engloba a todas esas empresas que utilizan medios sociales y necesitan de estos algoritmos para solventar abusos en sus medios sociales. Sin la necesidad de tener a alguien baneando dichos comentarios de manera manual. Ya que el proceso es muy lento si se realiza de manera manual. Un ejemplo son los comentarios realizados en Twitter o Facebook e incluso en los propios chats en línea como pueden ser el de los videojuegos.

A modo conclusión, es que los mensajes sean clasificados previamente antes de ser posteados en dicho medio social. Aunque no todas las plataformas pueden solventar este problema, lo que acaba pasando es que estos mensajes son procesados a posteriori y son evaluados. Un ejemplo es el sistema de baneo en el juego en línea League of Legends, el cual a finales del 2016 obtuvo más de 100 millones de jugadores activos mensualmente y aún continúan creciendo año tras año. Estos jugadores pueden o bien mientras están esperando a que empiece la partida o durante esta, chatear entre ellos. No obstante, algunos jugadores aprovechan este chat para abusar de otros jugadores, teniendo una conducta antideportiva.

En 2017 puso en marcha su algoritmo de detección de abusos, el cual consistía valorar los comportamientos de los distintos usuarios. En caso de encontrar actitudes antideportivas establecieron que como consecuencia dicho jugador debía ser baneado, y por lo tanto se le imposibilitaba jugar durante un periodo de tiempo. Cabe decir que actualmente siguen utilizando dicho algoritmo.

Sin embargo, en el blog se mencionó que todavía no es perfecto. Por ello los jugadores tienen la posibilidad de reclamar en caso de que consideren que han sido baneados de manera injusta.

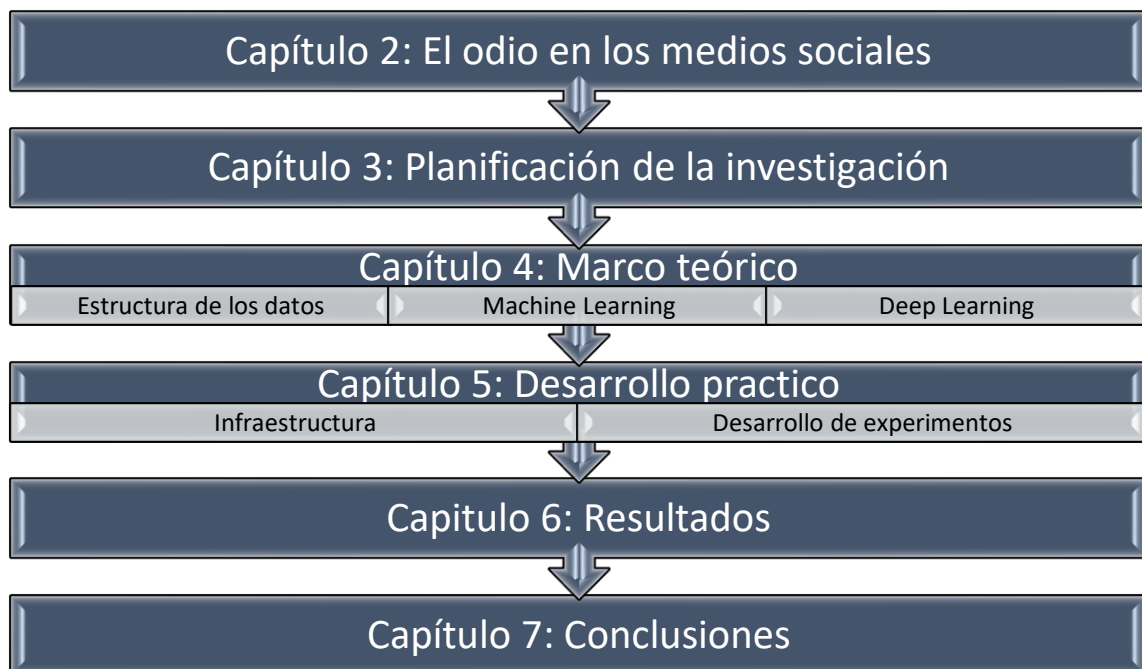
## 1.5. Estructura del proyecto

La estructura del proyecto ha sido diseñada de forma que, la primera parte es una introducción a como los medios sociales han sido afectados por la sociedad, los cuales han sido afectados por creación de contenido de odio de los propios usuarios, y por ello se busca un proceso el cual resuelva este problema. A continuación, en el segundo punto se detalla más concretamente sobre el odio generado en estas, aunque se centra más concretamente en la afectación a las redes sociales. E intentando ver como la IA puede dar una solución a dicho problema.

En el tercer apartado se mostrará la planificación estimada total del proyecto de investigación a realizar.

En el cuarto apartado, se mostrará el marco teórico, para poder entender todas las aplicaciones prácticas realizadas. Y en el quinto apartado se verá la configuración del entorno junto a los experimentos realizados.

Finalmente, en el sexto apartado se desarrollarán las conclusiones a partir de los experimentos realizados.



## 2.El odio en los medios sociales

Existen varios filósofos que hablan y definen el odio, uno de los más importantes es Empédocles, el cual definía el odio de dos maneras distintas. En primer lugar, una inclinación o rechazo con respecto a lo que sentimos. En segundo lugar, donde se expresa el origen de lo que, propiamente, se une o desune de la naturaleza.

Tal y como menciona Empédocles en su primera definición el rechazo que se genera con respecto a lo que sentimos, es un sentimiento propio de la humanidad, el cual siempre está ahí.

El odio es una emoción muy negativa, poderosa y altamente peligrosa, muchas investigaciones provenientes de la psicología social y clínica afirman que el odio posee una gran capacidad de aprendizaje y contagio. Funciona como una bomba fácil de activar y difícil de controlar. Antiguamente este odio era transmitido como mucho a las personas más cercanas de la persona que lo generaba y según el poder de convicción con el que lo transmitía llegaba a generar un mayor o menor daño.

Pero es aquí en el momento en el que encontramos los medios sociales. Unas herramientas que permiten a los usuarios opinar y comentar sobre cualquier publicación o generar mensajes con contenido propio, una de las ventajas de las redes sociales es la anonimidad de publicación de mensajes pudiendo utilizar un apodo, para referirte dentro de cualquier red social. Aunque como bien sabemos, no todos estos mensajes que se producen van dirigidos a generar un bien o un comentario imparcial, muchos de estos se aprovechan para generar odio. Es así que quien le ese mensaje, puede verse ofendido o generar un estado de ira por el propio contenido de este. Cabe mencionar, que este tipo de mensajes pueden ser con contenidos totalmente diferentes, desde mensajes sexistas, racistas, religiosos, anti sistémicos, de entre otros muchos.

Aquí es cuando entran dos perfiles de personas que generan este tipo de mensajes. En primer lugar, encontramos al denominado perfil Troll, el cual se ocultan mediante perfiles falsos, intentando incordiar, para generar confrontaciones y polémicas. Haciendo comentarios sobre las distintas publicaciones dentro de cualquier medio social en el cual puedan entrar, un ejemplo sería un usuario crea un Tweet y este lo comenta con algún mensaje confortativo por tal que el usuario que ha creado el Tweet le conteste y empiece una polémica.

En segundo lugar, encontramos el perfil del Hater, este el objetivo es llevar el odio generado online, al entorno offline. Intentando generar consecuencias en la vida real. De estos hay varios tipos, desde los que esperan que algún usuario cometa un error y lo empiezan a acribillar en estos medios por tal de que se vea perjudicado totalmente sin motivo alguno directo, hasta aquellos meticulosos que generan tales mensajes para poner la población en contra y afecten a la vida de un grupo reducido de personas a las cuales va dirigido.

Estos dos perfiles de usuarios cada vez más notorios en las redes sociales generan un gran problema en la sociedad, ya que afecta a las personas individualmente. Por ello muchas entidades intentan solventar de una manera eficiente este tipo de actos. Por un lado, se encuentran personas controlando dichos los medios sociales para dar con este tipo de usuarios

y sancionarlos, la realización de esta tarea es muy manual ya que un usuario debe de estar constantemente revisando si un comentario es positivo o negativo. Además, que el proceso es lento, por lo que analizar cada uno de los comentarios puede llegar a ser mortal, un ejemplo es Twitter que genera aproximadamente 500 millones de tweets al día. Por otro lado, se está llevando al uso de la Inteligencia Artificial, capaz de clasificar millones de comentarios en segundos, teniendo la capacidad de decisión en discernir entre contenidos que sean positivos o negativos.

Aquí entra la investigación realizada, con la finalidad si es capaz la inteligencia artificial llegar a aprender que mensajes son ofensivos. Indicando además que tipo de mensaje ofensivo es. Esto permite al medio social controlar la manera en que estos mensajes sean publicados o filtrados. Con el fin de evitar avivar la llama de la ira en los diferentes usuarios, por culpa de este tipo de comentarios.

### 3. Planificación de la investigación

En este apartado se muestra la planificación y organización del proyecto. Para ello se ha utilizado una herramienta llamada diagrama de Gantt. Esta ha permitido de forma gráfica exponer el tiempo dedicado para cada una de las tareas. Cabe decir, que por un lado encontramos el tiempo ideal y por otro la ejecución real del proyecto.

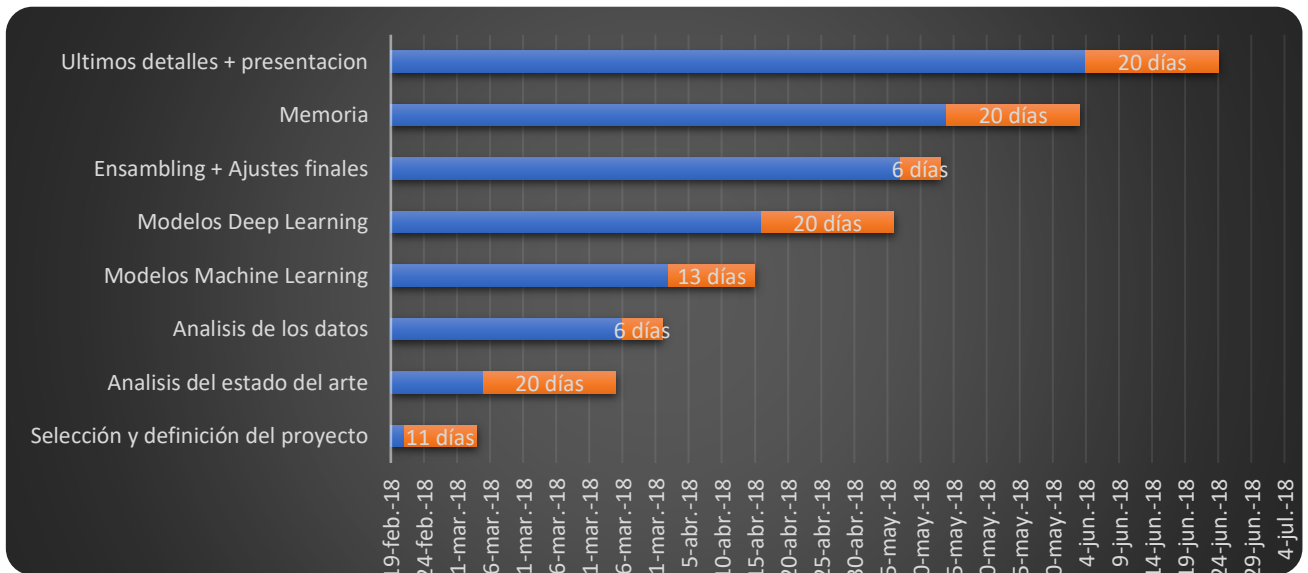


Figura 8 Diagrama de Gantt planificación ideal

Así pues, como se puede observar en la Figura 8 el tiempo ideal determinado para la selección y definición del proyecto, tenía una durabilidad de 11 días. Puesto que, era necesario que se plantearan que herramientas se usarían y como se llevaría a cabo la investigación.

Seguidamente, en Análisis del estado del arte se estableció un periodo 20 días donde investigaron proyectos relacionados tanto con el procesamiento del leguaje como la clasificación de textos. Una vez realizado lo anterior, se procedió a analizar los datos que se disponían de la competición de Kaggle en un periodo de 6 días. A continuación, se planteó realizar distintos modelos de Machine Learning para la clasificación de textos con una duración de 2 semanas. Todo ello planteando, en la ejecución de estos modelos y adaptarlos considerablemente. Posteriormente se realizaría la misma ejecución que con los algoritmos de Machine Learning, pero esta vez creando algoritmos de Deep Learning, ya que es necesaria realizar investigaciones sobre la gestión de capas de cada algoritmo se previó una duración de 20 días.

Para finalizar la parte experimental, se procedió al uso de algoritmos de ensembling y se realizarían los ajustes pertinentes a los algoritmos anteriores, el periodo total seria de 6 días.

Y ya para finalizar en referencia a la memoria, al ser un apartado extenso en el que es necesario plasmar estructuradamente todos los procesos e ideas del proyecto se determinó un tiempo de 20 días. Y para la preparación de la presentación y últimos retoques se planificó que con 20 días según la fecha oficial de presentación del proyecto.



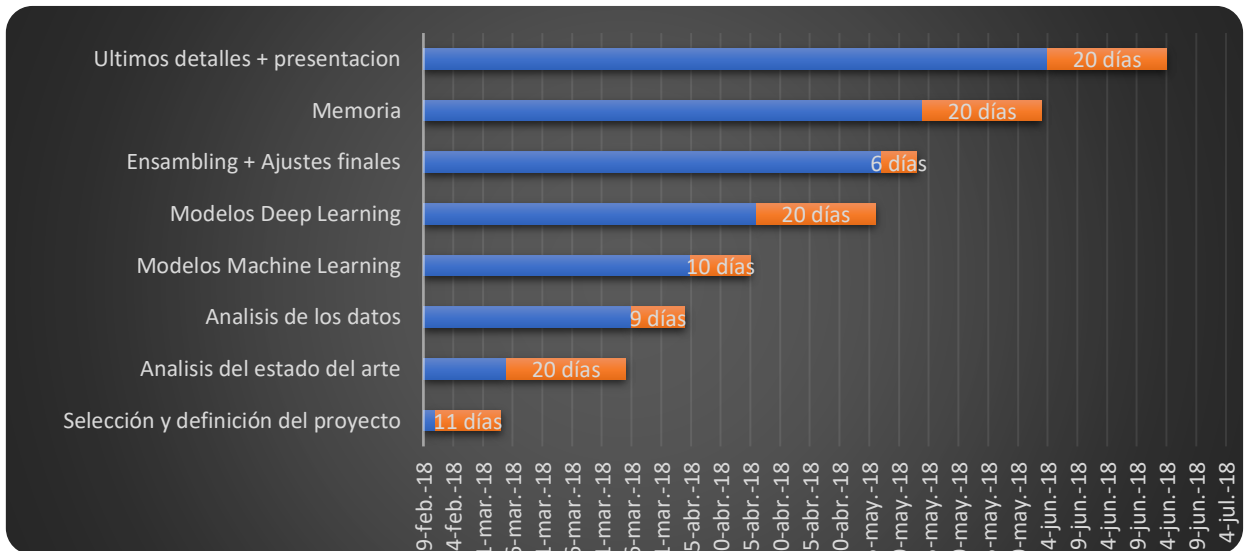


Figura 9 Diagrama de Gantt planificación realizada

No obstante, aunque anteriormente se han argumentado los periodos de tiempos ideales para la ejecución del trabajo. En realidad, la distribución del tiempo fue algo distinta como bien se muestra en la Figura 9.

Para la realización de los modelos de Machine Learning, cuando se estaban generando los vocabularios, no acaban de encajar. Por ello, los resultados eran considerablemente bajos. Así pues, se continuó investigado hasta llegar a dar con unos vocabularios que permitían mejorar el aprendizaje y predicción, el periodo necesario fue de aproximadamente 3 semanas. Estos fueron también utilizados en Deep Learning y por ello, no se redujo el tiempo a 2 semanas.

## 4. Marco teórico

A continuación, se mostrarán los diferentes algoritmos utilizados, para una mejor explicación este apartado está dividido en cuatro subapartados, estructuras de las palabras, algoritmos de Machine Learning, algoritmos de Deep Learning y Ensembling.

### 4.1. Estructura de los textos

Los textos son un tipo de dato no estructurado, por ello para poder ser tratados es necesario darles una estructura. Con la finalidad de que puedan ser tratados como valores o vectores numéricos. Este es el punto más importante antes de aplicar un algoritmo de clasificación. Según la estructura establecida los algoritmos serán capaces de aprender y clasificar mejor. En este caso se mostrarán tres distintas formas de representar el conjunto de textos o documentos.

#### 4.1.1. Matriz término documento

Es el algoritmo de representación de textos más simple, este algoritmo se basa en la selección de N palabras que están incluidas en los diferentes textos que se desean analizar.

Cada texto lo denominaremos documento, y el conjunto de documentos a analizar lo denominaremos corpus y estos serán representados a partir de vectores.

Las palabras pueden ser elegidas por ejemplo según la cantidad de veces que aparecen en el corpus. Una vez obtenidas, estas pasaran a ser las columnas de los vectores. Para esta selección de palabras la denominaremos vocabulario y cada palabra será denominada característica.

A partir de ahora cada vector, que se genere tendrá como tamaño el número de palabras del vocabulario.

Supongamos que de corpus disponemos de:

- El gato esta tumbado en el tejado de una casa.
- El perro duerme en la casa de enfrente.
- El perro gruñe siempre al mismo gato.

Un posible vocabulario sería contar el número total de palabras que aparecen en el corpus y coger las que aparezcan más de x veces, en este caso escogeremos  $x > 2$ .

el	gato	esta	tumbado	en	tejado	de	una	casa
4	2	1	1	2	1	2	1	2
perro	duerme	la	enfrente	gruñe	siempre	mismo	al	
2	1	1	1	1	1	1	1	

Por ello el vocabulario se verá formado por [el, gato, en, de, casa, perro].

Una vez se tiene el vocabulario se procederá a la creación de la estructura, la estructura final es una matriz  $n \times m$  donde n es el número filas, que debe de ser igual al número de documentos en

el corpus, en el ejemplo anterior es igual a 3. Y m es el número de columnas, el cual viene definido por la cantidad de palabras del vocabulario, en este caso 6.

El orden del vocabulario es indiferente, lo importante es siempre mantener el mismo orden.

Para la representación de cada uno de los vectores se incluirá en cada posición  $n_i, m_j$  la cantidad de veces que aparece la palabra en la columna j en el texto i.

En el caso de  $i = 1, j = 1$  se tendría la fila 1 con la palabra "el" la cual aparece en la fila 1 un total de 2 veces.

2					

Procediendo a rellenar todos los vectores el resultado final sería.

Vocabulario:

el	gato	en	de	casa	perro
----	------	----	----	------	-------

Matriz término documento:

2	1	1	1	1	0
1	0	1	1	1	1
1	1	0	0	0	1

### 4.1.2. TF-IDF vector

El algoritmo TF-IDF es un algoritmo basado en la frecuencia, tal y como indica su nombre completo en inglés “Term frequency – Inverse document frequency”, esto viene a decir la frecuencia de ocurrencia del término en la colección de documentos.

Este algoritmo se basa en maximizar aquellas palabras que aparecen muchas veces en un único documento, pero aquellas que en la mayoría de documentos suelen aparecer se consideran no importantes, un ejemplo de dichas palabras son los artículos. Permitiendo obtener un vocabulario capaz realmente de representar cada uno de los documentos del corpus. Este tipo de selección de características se puede aplicar a la selección de palabras de Bag Of Words.

Supongamos unos documentos donde tenemos su término (palabra) y la cantidad de veces que aparece en cada documento:

Documento 1		Documento 2		Documento 3	
el	2	el	3	el	1
gato	4	perro	2	perro	1
esta	2	duerme	1	gruñe	1
feliz	2	en	1	siempre	1
no	1	coche	1	al	1
para	1	casa	1	mismo	1
				gato	1
Total	12		9		7

Posteriormente procederemos al cálculo del TF- IDF:

En primer lugar, el cálculo del TF se basa en el número que cada termino aparece en un documento dividido por el número total de términos en dicho documento.

$$TF = (término, documento) = \frac{n^{\circ} \text{ que aparece en documento}}{\text{total de terminos del documento}} \\ = 0.1667$$

Ejemplo de TF:

$$TF(\text{el}, \text{Documento 1}) = \frac{2}{12} = 0.1667$$

$$TF(\text{el}, \text{Documento 2}) = \frac{3}{9} = 0.3333$$

$$TF(\text{el}, \text{Documento 3}) = \frac{1}{7} = 0.1429$$

Este valor principalmente ve reflejado para cada documento que términos son frecuentes. Para completarlo se debería realizar sobre cada uno de los términos.

El siguiente calculo a realizar es el IDF.

$$IDF(término) = \log\left(\frac{N}{n}\right)$$

Donde N es el número total de documentos en el corpus, en este caso 3 y n es la suma de únicamente si el termino aparece o no en cada uno de los documentos, es decir el valor máximo que puede alcanzar n es N.

Ejemplo de IDF:

$$IDF(el) = \log\left(\frac{3}{3}\right) = 0$$

$$IDF(gato) = \log\left(\frac{3}{2}\right) = 0.1761$$

$$IDF(perro) = \log\left(\frac{3}{2}\right) = 0.1761$$

Finalmente, para calcular el TF-IDF se deben multiplicar ambos términos

$$TF - IDF(\text{término}, \text{documento}) = TF \times IDF$$

Ejemplo de TF - IDF:

$$TF - IDF(el, \text{Documento 1}) = 0.1667 \times 0 = 0$$

$$TF - IDF(gato, \text{Documento 1}) = 0.3333 \times 0.1761 = 0.0587$$

Con este último ejemplo se puede ver cómo ha penalizado el término “el” en el documento 1 y en cambio “gato” tiene una mejor valoración.

La matriz final es una matriz donde cada columna es un término único, y cada fila es cada uno de los documentos. En cada posición i, j se verá asignado el cálculo pertinente al TF-IDF en caso que el documento no disponga del término del vocabulario a analizar este directamente será valorado a 0.

El vocabulario no importa el orden en el que se genera, lo importante es mantener siempre la misma estructura del vocabulario.

Vocabulario = [el, gato, esta, feliz, no, para, perro, duerme, en, gruñe, mismo, siempre, coche, al, casa]

<b>Vocabulario</b>	<b>El</b>	<b>Gato</b>	<b>Esta</b>	<b>Feliz</b>	<b>No</b>
<b>Documento1</b>	0	0.0587	0.0795	0.0795	0.0398
<b>Documento2</b>	0	0	0	0	0
<b>Documento3</b>	0	0.0252	0	0	0
<b>Vocabulario</b>	<b>Para</b>	<b>Perro</b>	<b>Duerme</b>	<b>En</b>	<b>Gruñe</b>
<b>Documento1</b>	0.0398	0	0	0	0
<b>Documento2</b>	0	0.3913	0.0530	0.0530	0
<b>Documento3</b>	0	0.0252	0	0	0.0682
<b>Vocabulario</b>	<b>mismo</b>	<b>Siempre</b>	<b>Coche</b>	<b>Al</b>	<b>Casa</b>
<b>Documento1</b>	0	0	0	0	0
<b>Documento2</b>	0	0	0.0530	0	0.0530
<b>Documento3</b>	0.0682	0.0682	0	0.0682	0

### 4.1.3. Tokenización de las palabras

El algoritmo de tokenización se basa primeramente en separar los documentos en vectores teniendo cada documento como vectores de términos. Una vez realizado este proceso se calcula la frecuencia de aparición de cada término.

Supongamos que el corpus que disponemos es:

- El gato esta tumbado en el tejado de una casa.
- El perro duerme en la casa de enfrente.
- El perro gruñe siempre al mismo gato.

Con la tabla de aparición de términos:

el	gato	esta	tumbado	En	tejado	de	una	casa
4	2	1	1	2	1	2	1	2
perro	duerme	la	enfrente	gruñe	siempre	mismo	al	
2	1	1	1	1	1	1	1	

Una vez calculada se obtienen los N términos con mayor frecuencia de aparición.

La selección de N va en función del número de documentos y palabras que se dispongan en el corpus. Este será el vocabulario a utilizar, hay que tener en cuenta que contra mayor es el vocabulario la realización de tareas de Clasificación puede verse drásticamente afectado, al igual que al consumo de memoria requerido.

Siguiendo el ejemplo anterior indicamos como N los Top 5. Imaginemos que el orden que nos ha dado por frecuencias es el siguiente:

[el, gato, perro, en, casa, de, duerme, gruñe, siempre, mismo, ...]

Las características que se tendrán en el vocabulario son las siguientes:

[el, gato, perro, en, casa]

Una vez se han seleccionado las N características que formarán parte del vocabulario, a estas se les asignará un identificador único. Para el uso de este vocabulario la mejor forma de representarlo es en un diccionario clave - valor.

En este caso el vocabulario quedaría de la siguiente forma:

{el : 1, gato : 2, perro : 3, en : 4, casa : 5}

A continuación, por cada documento se generará un vector, este vector de cada documento tendrá una dimensionalidad diferente según la cantidad de términos que coincidan con las características del vocabulario. Para generar cada vector de cada uno de los documentos se le irá añadiendo una dimensionalidad nueva con el número de identificador asignado por el

vocabulario, en caso que la palabra no aparezca en el vocabulario, esta no se añadirá a dicho vector. Este proceso se llama pasar los documentos a secuencias.

Documento 1: [1, 2, 4, 5]

Documento 2: [1, 3, 4, 5]

Documento 3: [1, 3, 2]

Para completar y finalizar la estructura de Tokenización, se deben acolchar las secuencias previas, para así generar una matriz  $n \times m$  donde  $n$  son las filas que corresponden al número de documentos y  $m$  es un valor alto que corresponde a las columnas, este valor debe ser superior al tamaño del vector con mayor dimensión. En este caso supongamos que es 6. Entonces, por cada fila se irán añadiendo 0 hasta que el tamaño de la secuencia sea capaz de rellenar los valores restantes.

Es decir, la matriz final en el ejemplo anterior quedaría:

$$M = \begin{pmatrix} 0 & 0 & 1 & 2 & 4 & 5 \\ 0 & 0 & 0 & 3 & 4 & 5 \\ 0 & 0 & 0 & 1 & 3 & 2 \end{pmatrix}$$

#### 4.1.4. Word2vec

Es una red neuronal de dos capas, el objetivo es mediante un corpus, extraer las características y pasarlas a vectores. Para representar una palabra en vector es realizado mediante una serie de documentos pre entrenados los cuales se han detectado las similitudes y diferencias entre las palabras. La ventaja de hacer uso de word2vec es la realización de funciones sobre vectores. Pudiendo así hacer operaciones como  $\text{king} - \text{man} + \text{woman} = \text{queen}$ , equivalente a la suma y resta de vectores de la Figura 10.

La utilización para predecir clases mediante word2vec es únicamente cuando se utilizan algoritmos de Deep Learning. Los cuales se basan en optimizar los pesos de cada característica del corpus. Para así mejorar el aprendizaje.

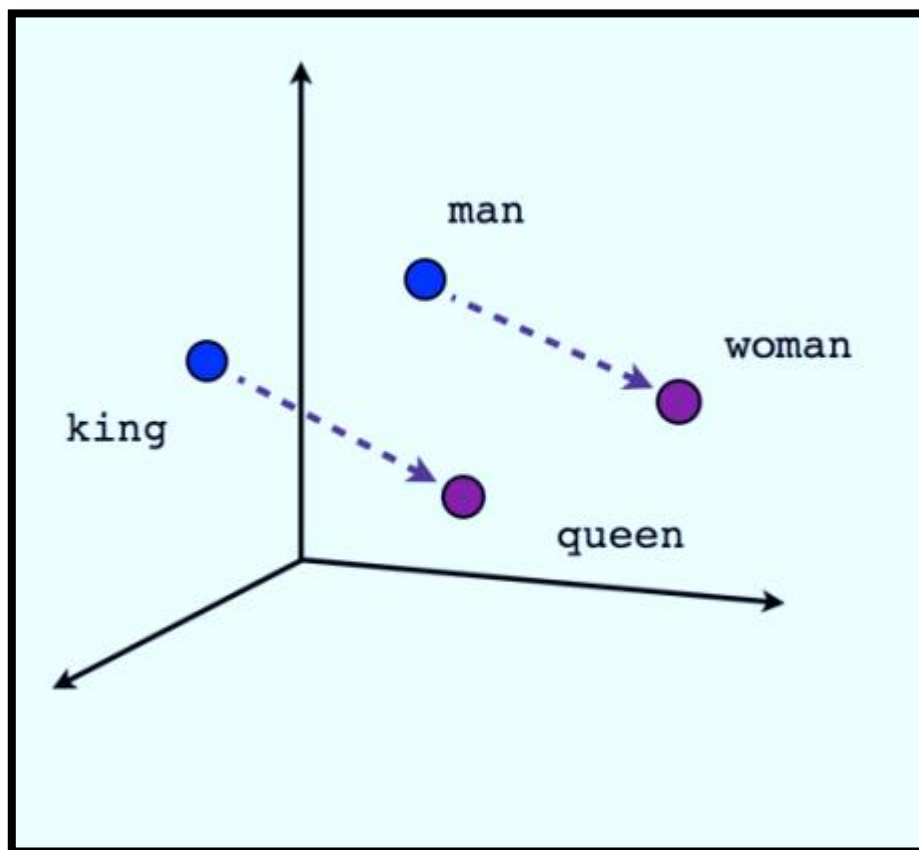


Figura 10 Representación Word2vec



## 4.2. Machine Learning

El Machine Learning o Aprendizaje Automático es un subcampo de la Inteligencia artificial, el objetivo principal es hacer que una máquina, en este caso un ordenador sea capaz de aprender sobre un modelo, y este tenga la capacidad de poder tomar decisiones según el aprendizaje que le sea dado.

### 4.2.1. Tipos de aprendizaje

Los algoritmos de Machine Learning se clasifican de dos tipos distintos según su aprendizaje. Si este es ser supervisado o no supervisado.

#### 4.2.1.1 Aprendizaje supervisado

Denominamos un aprendizaje supervisado cuando el modelo de aprendizaje tiene su solución, este tipo de algoritmos son utilizados para encontrar patrones generados por el pasado y posteriormente clasificar los nuevos que no han sido previamente clasificados.

Ejemplo de aprendizaje supervisado:

Se tienen almacenadas múltiples informaciones de diferentes hojas de flores, las cuales hacen referencia a la flor a la que pertenecen, podría ser color, tamaño de largo y ancho, forma, etc. Y a partir de toda la información que se tiene almacenada introduciendo una flor nueva las cuales han sido tomadas las distintas medias de las características el algoritmo sería capaz de indicar que tipo de flor es.

#### 4.2.1.2 Aprendizaje no supervisado

Por otro lado, encontramos los de aprendizaje no supervisado, estos también utilizan datos históricos, pero con la variante que no disponen de la solución final, es decir el clasificador no sabe que está clasificando, el objetivo de este tipo de clasificación es encontrar patrones o tendencias no fácilmente visibles a simple vista, pero son interesantes de que sean agrupados.

Ejemplo de aprendizaje no supervisado:

Se le pasa información de una red social para encontrar subgrupos en esta y poderlos utilizar en campañas, como por ejemplo de márketing. Así, encontrando los patrones y tendencias de los usuarios es capaz de introducir un nuevo usuario y asignarle una categoría según el grupo o patrón encontrados.

### 4.2.2. Problemas de clasificación

En la investigación a realizar se centrará de los algoritmos de Machine Learning de aprendizaje supervisado, por ello cabe destacar los distintos tipos de problemas de clasificación que se pueden llevar a cabo.

#### 4.2.2.1 Clasificadores binarios

En primer lugar, encontramos los clasificadores mono clase o clasificación binaria.

Los clasificadores binarios son utilizados cuando se desea discernir entre dos tipos. Un ejemplo de clasificación binaria sería distinguir si un mensaje es Spam o es No Spam. A estos se les un

0 para todo el contenido que hace referencia a una clase y un 1 a los que pertenecen a la contraria, es decir todos los mensajes de entrenamiento que No son de Spam tienen un valor de clase 0 y los que son de Spam tienen un 1.

Un ejemplo de dataset de Spam sería:

Remitente	Asunto	Mensaje	Clase	Clase Binaria
<b>soyunapersonareal@gmail.com</b>	Hola	Este es un mensaje valido	No spam	0
<b>vendodedtodo@nombredesconocido.com</b>	****	Este mensaje te vende de todo	Spam	1
...	...	...	...	
<b>perfilfalso@info.com</b>	****	* * * * * * * * * * * * * * * * *	Spam	1

#### 4.2.2.2 Clasificadores multiclase

Por otro lado, se encuentran los problemas de clasificación multiclase, este tipo de problemas viene definido a que un mismo contenido puede pertenecer o no a varias clases distintas al mismo tiempo, el método más utilizado para tratar las distintas clases asignadas es la transformación binaria. Esta se realiza mediante tres pasos.

El primer paso a realizar es asignar a cada clase un valor numérico y único. No puede haber dos clases con el mismo valor.

El segundo paso a realizar es utilizar estos valores en vez de los nombres asignados a cada clase, para facilitar la búsqueda e indexación de cada una de las clases.

El tercer paso a realizar es la binarización de asignación de clases, es decir, transformar las múltiples clases en un número binario de longitud N, donde N es la cantidad de clases disponibles, para ello las clases serán asignadas a una posición, las cuales nunca deben de ser cambiadas, un posible orden de estas es por orden ascendente o descendiente.

Ejemplo de clasificación multiclase:

Se desea generar un clasificador de películas el cual introduciendo características de una película nueva índice a que categorías pertenece.

Supongamos que las categorías a clasificar son Terror, Romance, Drama, Thriller un total de 4 categorías.

En este caso disponemos de la siguiente información:

Película	Terror	Romance	Ciencia ficción	Thriller
IT	Si	No	No	Si
La forma del agua	No	Si	No	Si
...	...	...	...	...
El señor de los anillos	No	No	Si	No
Star wars	No	No	Si	No

Paso 1:

Se asigna a Terror = 1, Romance = 2, Ciencia ficción = 3, Thriller = 4

Paso 2:

Película	Clases
IT	1,4
La forma del agua	2,4
...	...
El señor de los anillos	3
Star wars	3

Paso 3 manteniendo el orden ascendente de los valores es decir [Terror, Romance, Ciencia ficción, Thriller]

Película	Clases
IT	1001
La forma del agua	0101
...	...
El señor de los anillos	0010
Star wars	0010

Una vez realizada la transformación de las distintas clases a binario los algoritmos pueden tratar de dar la solución de haciendo uso de distintas estrategias, según el algoritmo que se desea utilizar y el tipo de problema a tratar, es decir si es de clasificación o regresión.

Los más utilizados son uno contra el resto y uno contra uno.

#### 4.2.2.2.1. Uno contra el resto (One vs The Rest)

Este tipo de estrategias se basan en generar un clasificador por clase y este aprende haciendo uso de su clase asignada frente al resto de clases, la ventaja de utilizar esta estrategia es su eficiencia, ya que con tan solo N clasificadores es suficiente y su interpretación es muy fácil.

Gracias a este sistema es fácilmente crear un seguimiento sobre que aprendizaje se ha realizado en cada clase. Esta estrategia es la más utilizada por lo mencionado anteriormente.

#### 4.2.2.2.2. Uno contra uno (One vs One)

Esta estrategia se basa en la construcción de un clasificador por cada par de clases, es decir, se tendrán un total de  $n\_classes * (n\_classes - 1) / 2$  clasificadores. Una vez realizados todos los clasificadores y entrenados a la hora de predecir estos lo hacen por votación.

Supongamos como ejemplo un problema de 4 clases el cual la clase 1 predice sobre algún termino, este compara que resultado ha dado cada uno de los clasificadores de cada clase:

Clase 1 vs Clase 2 = Tipo 1

Clase 1 vs Clase 3 = Tipo 3

Clase 1 vs Clase 4 = Tipo 1

La predicción sobre la clase 1 es 2 votos vs 3 por lo que por mayoría indicara que sobre la predicción tiene un  $2/3 * 100 = 66.667\%$  de pertenecer a la clase 1.

Este cálculo se realizaría con cada una de las combinaciones posibles.

Como se puede observar, este tipo de clasificación es más lenta que la anterior, en los casos de tener muchas clases puede llegar a generar un coste computacional muy alto, ya que se deben de entrenar una gran cantidad de clasificadores.

### 4.2.3. Clasificadores

Los clasificadores de Machine Learning que se mostrarán a continuación son los utilizados en la parte experimental, ya que existen una gran cantidad de clasificadores de Machine Learning, pero al tener un problema multiclase se han seleccionado los más adecuados para solucionar este problema, en este caso encontraremos los algoritmos de Regresión Logística, Máquinas de soporte vectorial con un kernel lineal, Naive Bayes, Stochastic Gradient Descent, Árbol de Decisión (Decision Tree), todos ellos han sido utilizados mediante la estrategia uno contra el resto (One vs the rest)

#### 4.2.3.1 Regresión Logística (Logistic Regression)

La regresión logística es un método utilizado para problemas de clasificación binaria, el nombre viene principalmente de su función primaria la función logística o también llamada función sigmoide (Brownlee, J. 2016).

La función sigmoide tiene la particularidad de poner cualquier valor real entre los valores 0 y 1.

La función matemática es:  $f(X) = 1/(1 + e^{-x})$

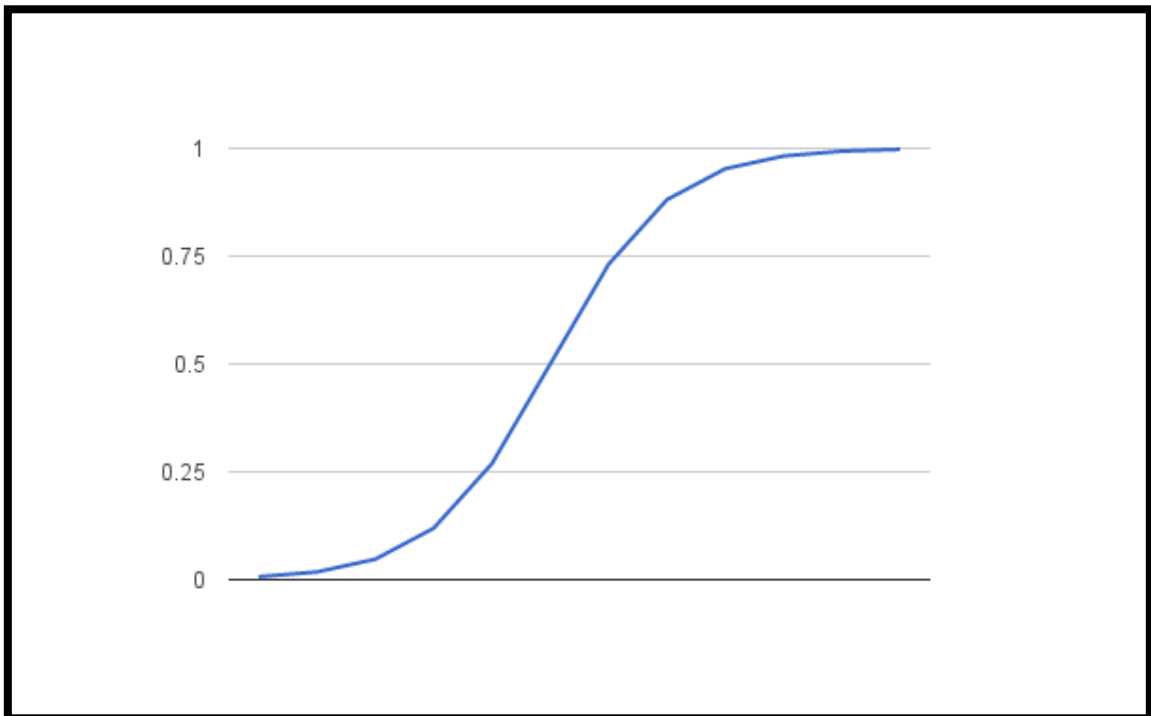


Figura 11 Función Sigmoide

Siguiendo el concepto de la función sigmoide, la regresión logística cuya funcionalidad es encontrar las probabilidades que dado un valor este corresponda a una categoría. Se ha modificado de la siguiente manera:

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

En esta función  $\beta_0, \beta_1, \dots, \beta_p$  son desconocidas, y son las que se determinarán según los datos de entrenamiento.  $\beta_0$  es equivalente al coeficiente de intercepción y  $\beta_1, \dots, \beta_p$  corresponde al coeficiente de cada uno de los predictores. Para conseguir estos valores, se utiliza la estimación por máxima similitud (maximum likelihood estimation).

Esta consiste en la maximización de cada uno de los parámetros  $\beta$  que corresponde a cada una de las clases. La fórmula para calcular cada una de las  $\beta$  es:

$$\hat{\beta} = \operatorname{argmax}_{\beta} \sum_{i=1}^n \log f_X(x_i; \beta)$$

Donde  $x_i$  corresponde a cada una de las muestras correspondientes a la clase de  $\beta$ .

Una vez realizado el cálculo de cada una de las betas, se procederá a calcular la probabilidad de que dado una serie de valores  $X$  calcular la probabilidad de pertenecer a la clase  $k$ . Para hallar a que clase pertenece se debe realizar la formula anterior  $K - 1$  veces donde  $K$  es el total de clases.

$$Pr(Y = k | X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

Para calcular la clase restante tan solo es necesario realizar la resta de las probabilidades de cada una de las diferentes clases sobre 1 es decir:

$$Pr(Y = k_0 | X) = 1 - Pr(Y = k_1 | X) - Pr(Y = k_2 | X) - \dots - Pr(Y = k_n | X)$$

La selección de la clase a la que pertenece en caso de necesitar indicar la clase es la que la probabilidad sea la más alta.

#### 4.2.3.2 Máquinas de soporte vectorial con un kernel lineal (Linear Support Vector Machine)

El objetivo de este tipo de clasificación es mediante un hiperplano de separación diferenciar entre dos clases (Carmona Suárez, E. J. 2014). Para la generación de este hiperplano lo que hay que tener en cuenta es su maximización entre las diferentes clases más cercanas, cuando se habla de un problema multiclase se generan  $K$  hiperplanos que diferencian entre una clase y el resto, siempre maximizando la distancia entre ellas y el hiperplano por una constante  $C$ .

Un ejemplo de hiperplano sería el que encontramos en la Figura 12, donde los vectores de soporte son los rellenos de su propio color, por lo tanto el hiperplano óptimo es el que se genera a partir de los dos hiperplanos paralelos generados por los vectores de soporte, en este caso son los definidos por la línea de puntos y el vector óptimo es el generado por la línea continua que exactamente está a una distancia  $T$  de cada una de los hiperplanos de los vectores de soporte, distinguiendo en este caso entre la clase de Cuadrados y Círculos.

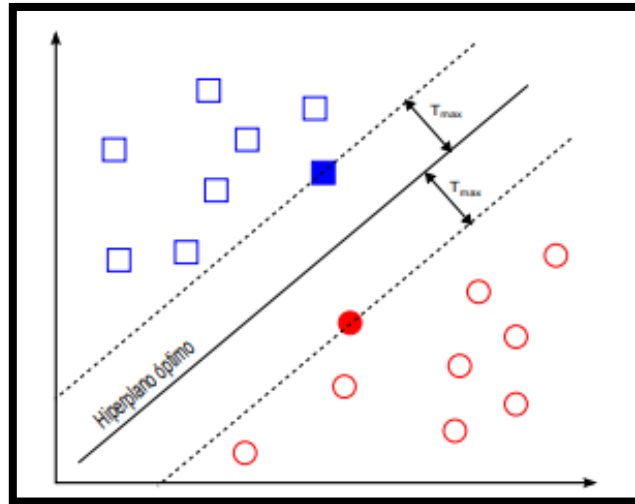


Figura 12 SVM hiperplano óptimo

El caso de la Figura 12 sería lo ideal que nos gustaría encontrar cuando tenemos varias clases distintas, pero la realidad no es así, en muchos casos se tienen datos con valores de ruido, es decir no son separables por una única línea.

Para poder generar el hiperplano óptimo en primer lugar se deben generar la función lineal para cada una de las clases teniendo en cuenta los vectores de soporte, que serán aquellos más próximos a la diferencia entre 2 clases las cuales su función claramente separará entre dos clases sin tener ruido en lo que se considera la clase clasificada. Una vez realizada esta operación se procederá a la realización de encontrar la función lineal que maximice la distancia entre ambos hiperplanos. Una vez se tienen la función óptima, aunque entre la distancia de función óptima a cada uno de los hiperplanos haya ruido, es decir en frontera generada por el hiperplano óptimo tenga valores no adecuados en cada uno de los lados ya que no han sido correctamente separables. Así pues, será necesario controlar cuales han sido los no separables, estos los denominaremos  $\mathcal{E}$ . Estos están contenidos entre los dos hiperespacios generados por los vectores de soporte.

$$f(\omega, \varepsilon) = f(\omega) + C \sum_{i=1}^n \varepsilon_i$$

Por ello a la función óptima será necesario añadirle una constante C, la cual será elegida arbitrariamente, para controlar los  $\mathcal{E}$ . Si C tiende a  $\infty$  se estaría considerando un caso que los vectores son perfectamente separables por lo que  $\varepsilon_i$  tendería a 0. En caso contrario si C es muy pequeña tendiendo a 0 se permitirían todos y cada uno de los ejemplos mal clasificados.

Este valor es muy importante a la hora de clasificar porque según en qué casos se puede llevar a realizar un sobre ajustamiento del clasificador y con este le daríamos su flexibilidad.

Hay que recordar que donde se encuentran los valores mal clasificados también encontramos que si están bien clasificados así pues se deben de valorar para la hora de elegir C.

### 4.2.3.3 Naive Bayes

El objetivo de Naive Bayes es calcular a partir de las probabilidades de cada una de las clases. Teniendo en cuenta las probabilidades de cada uno de los atributos con sus respectivas clases (Epp, R. 2018).

Este algoritmo necesita de las clases a predecir que las denominaremos “C” al conjunto de clases y “c” a la clase individual. Cada uno de los atributos denominados donde al conjunto lo llamaremos F y a cada individuo f.

En primer lugar, se deben calcular sobre los datos a analizar o entrenamiento, para cada clase “c” las medias de cada uno de los atributos juntamente con su varianza.

En segundo lugar, sobre una muestra obtenida, que contiene cada uno de los atributos de  $f_1, f_2, \dots, f_n$  se debe calcular para cada clase la probabilidad a posteriori que la muestra pertenezca a una clase entre la constante de normalización. Y la que el resultado sea mayor es la clase asignada a dicha muestra. Para ello se utilizarán las funciones:

$$clasifica\_muestra(f_1, f_2, \dots, f_n) = \underset{c}{\operatorname{argmax}} \left( \frac{p(C = c) \prod_{i=1}^n p(F_i = f_i | C = c)}{\text{constante de normalización}} \right)$$

$$\text{constante de normalización} = \sum p(C = c) \prod_{i=1}^n p(F_i = f_i | C = c)$$

La probabilidad de una clase es la cantidad de muestras que pertenece a una clase entre el total de muestras.

La probabilidad que un valor de un atributo f correspondiente al atributo x de una clase c es:

$$p(f | c) = \frac{1}{\sqrt{2\pi\sigma_{xc}^2}} e^{\frac{-(f - \bar{f}_x)^2}{2\sigma_{xc}^2}}$$

$$\sigma^2 = \text{varianza}$$

### 4.2.3.4 Árbol de decisión

El Árbol de decisión es un algoritmo basado en la creación de decisiones simples. Las cuales generan nodos que recorriendo cada una de las condiciones llevan hasta una hoja correspondiente a una clase.

Una de las ventajas de este algoritmo es su fácil interpretación y velocidad de predicción. Aunque no suele ser uno de los mejores algoritmos de aprendizaje supervisado a nivel de precisión. Cabe decir, que según los datos puede llegar a dar un mejor resultado que un algoritmo lineal.

Para la realización de este algoritmo se deben realizar condiciones que separen constantemente si un dato pertenece a una clase o no. Usualmente los arboles de decisión si una condición la cumple se sigue al nodo de la derecha en caso contrario se procede al izquierdo, siempre teniendo un recorrido de arriba hacia abajo.



Hay que tener en cuenta que contra mayor número de nodos de condiciones se vaya generando en profundidad, este generará más overfitting de los datos. Por otro lado, siempre las condiciones deben separar dos subgrupos que maximicen la máxima cantidad que una clase este separado. Nunca se genera una condición si esta no mejora el estado anterior.

Una cosa a tener en cuenta, es que cuando se tienen muchos datos de entrenamiento con varias clases se puede llegar a generar árboles de decisión muy extensos, por ello se establece una profundidad máxima el cual el árbol pueda llegar evitando el overfitting y su coste de creación.

## 4.2.4. Optimizadores

### 4.2.4.1 Stochastic Gradient Descent

Es un algoritmo de optimización, utilizando normalmente para encontrar los pesos o coeficientes para modelos lineales como por ejemplo son la regresión logística o Support Vector Machine. Este algoritmo funciona mediante la predicción sobre los propios datos de entrenamiento utilizando un algoritmo lineal previamente establecido. Por cada predicción realizada se calcula el error, con este error se actualiza el modelo actual de forma que posteriormente sea reducido. Con el objetivo de llegar a tener en la función lineal los parámetros más óptimos para predecir.

El algoritmo utilizado para calcular cada uno de los coeficientes llamado Gradient Descent es:

En primer lugar, se inicializa con un coeficiente de 0.0 o un valor aleatorio muy bajo.

En segundo lugar, se calcula el coste aplicando ese coeficiente, sobre el algoritmo seleccionado.

En tercer lugar, se calcula su derivada, para indicar la pendiente generada de la función y así poderse orientar y obtener un coste menor la próxima vez que movamos el coeficiente.

En cuarto lugar, se indica un coeficiente de aprendizaje, este coeficiente es arbitrario y marcará la velocidad en la que algoritmo optimizará la función. Siendo el controlador de cuanto pueden llegar a cambiar los coeficientes en cada actualización.

En quinto lugar, se calcula un nuevo coeficiente siendo:

$$\text{coeficiente} = \text{coeficiente} - (\text{derivada del coste} \cdot \text{velocidad de aprendizaje})$$

Este proceso se repite hasta que el coste de la función es muy cercano a 0.

Una vez se ha entendido como funciona el algoritmo Gradient Descent, este es aplicado de manera Stochastic (estocástica), el cual indica que se debe de realizar un Gradient Descent por cada muestra que se dispone en el modelo de entrenamiento, es decir un total de N veces será ejecutado. Para cada ejecución de Gradient Descent se realizará un modelo de entrenamiento nuevo, escogiendo valores del modelo de entrenamiento al azar, este puede ser del (80% - 90%) de los datos. Y será con el que aprenderá el modelo lineal establecido.

Esto puede generar en conjuntos de entrenamiento con muchos datos una gran lentitud de aprendizaje. Por ello este algoritmo muchas veces es realizado un total de N veces para obtener unos valores decentes sin llegar a los óptimos.

### 4.3. Deep Learning

El Deep Learning o aprendizaje profundo es un conjunto de algoritmos basados en Inteligencia Artificial, intentando modelar abstracciones de alto nivel a partir de redes neuronales.

Las redes neuronales intentan simular las neuronas de un cerebro humano, estas se componen de capas y neuronas. Cada capa contendrá un número  $N$  de neuronas, las cuales todas estarán relacionadas en las distintas capas, cada relación tendrá asignado un peso "w". Estas capas se dividen en tres grupos. La capa inicial es la denominada capa de entrada. Esta contendrá una neurona por cada dato de entrada. Por otro lado, está la última capa, denominada capa de salida, esta contendrá una neurona por cada estado o clase disponible. Finalmente, se encuentran las capas intermedias, estas son denominadas capas ocultas, según la ejecución que se desee realizar se necesitarán más o menos capas ocultas y una cantidad variable de neuronas. A excepción de en la capa de salida se incluirá una neurona adicional en cada capa, llamada bias que siempre estará activada a 1. Esta neurona nunca será conectada por ninguna neurona de la capa anterior.

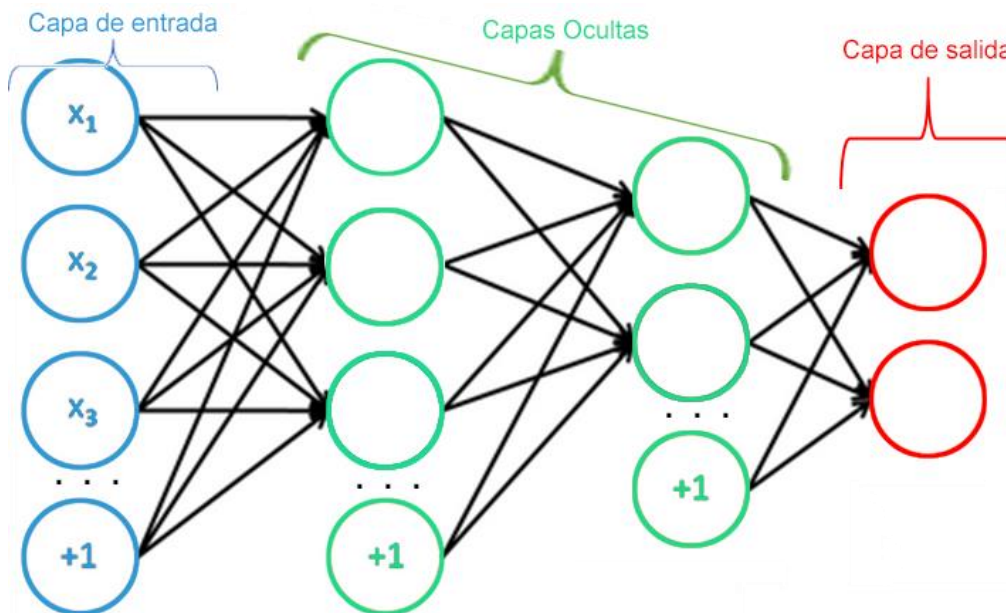


Figura 13 Red neuronal

Las neuronas estarán compuestas por una función activadora, según la función esta pasará de un estado activada o desactivada.

Usualmente para activar la neurona se utiliza una de las cinco funciones que se mostrarán a continuación:

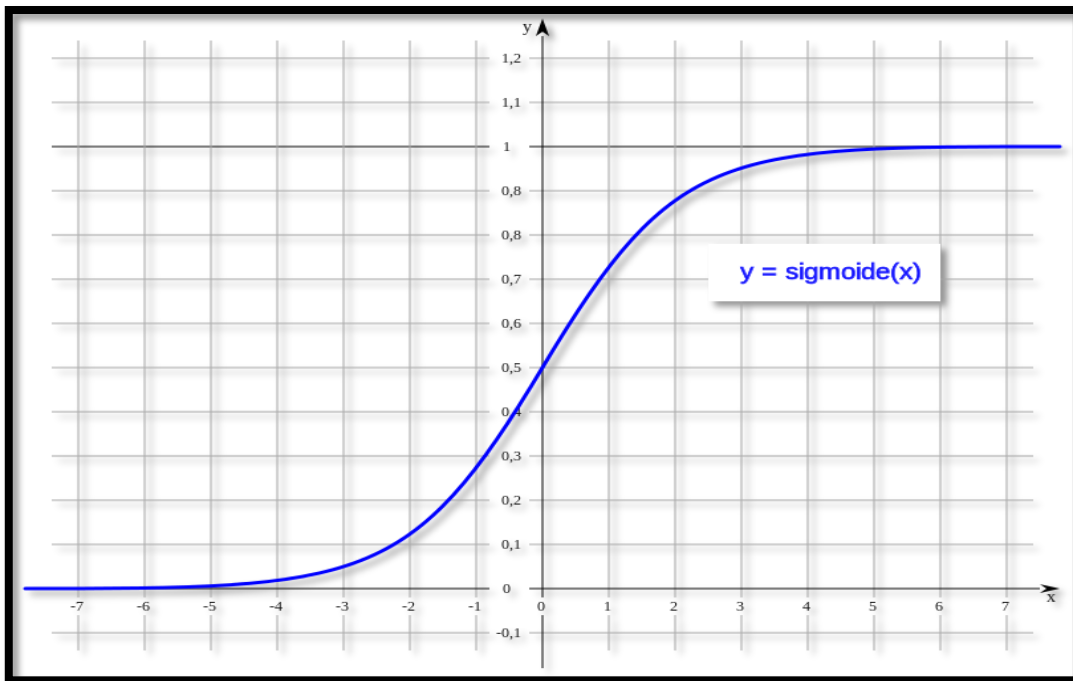
### **Función de neurona lineal:**

Esta función es uno de los modelos más simples para las neuronas. Donde  $b$  es el valor de la neurona bias y posteriormente se calcula la suma para cada uno de los valores de las neuronas conectadas a esta por el peso.

$$f(x) = b + \sum_i x_i w_i$$

### **Función sigmoide logarítmica:**

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



*Figura 14 Función Sigmoidal*

### **Función unidad lineal rectificada o rectified linear unit (ReLU):**

$$f(x) = \text{ReLU}(x) = \max(0, x)$$

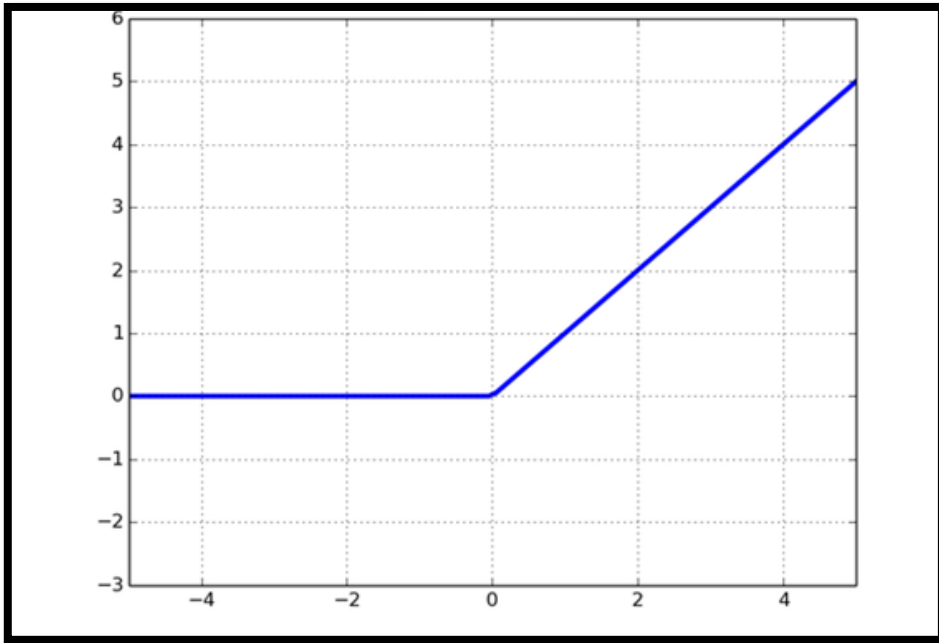


Figura 15 Función ReLU

**Función binaria o Binary threshold neurons:**

$$f(x) = \begin{cases} 1, & \text{si } x \geq 0 \\ 0, & \text{en caso contrario} \end{cases}$$

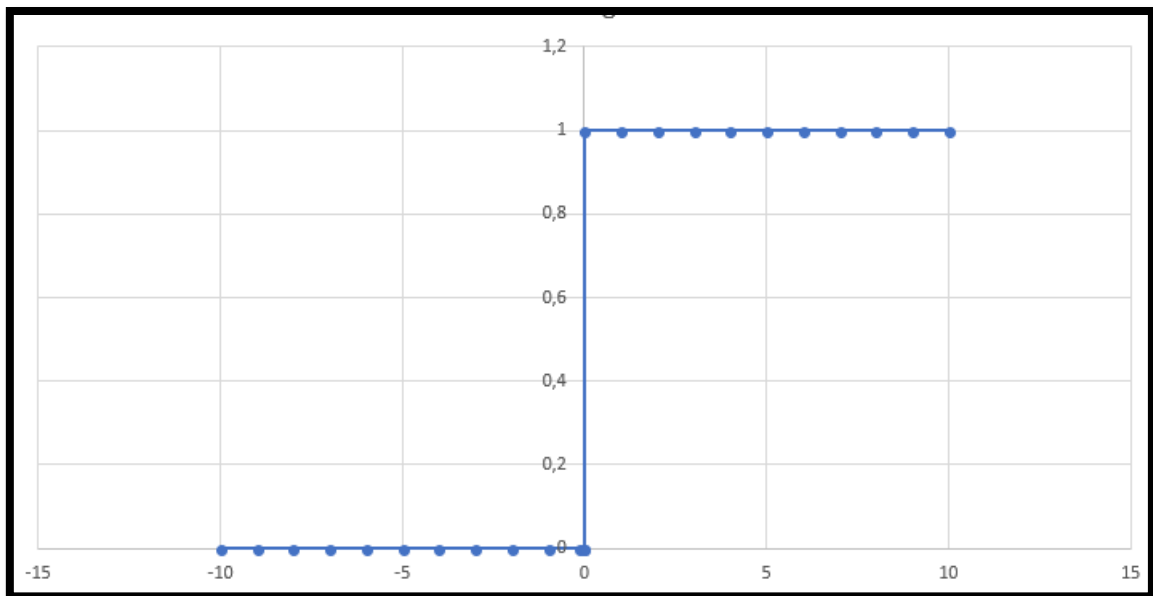


Figura 16 Binary threshold

### **Función Softmax:**

Esta función se utiliza principalmente para comprimir un vector  $x$  de dimensionalidad  $K$  a los rangos entre 0 y 1, es útil como función de activación para las capas de salida, ya que siempre mantiene la misma proporcionalidad del vector. Por ejemplo, el valor máximo siempre estará ubicado en la misma posición del vector solo que entre el rango 0 y 1.

$$f(\vec{x}) = \frac{e^{x_i}}{\sum_{i=1}^K e^{x_i}}$$

Usualmente, las neuronas combinan la función de neurona lineal aplicada a la función de activación. Es decir, en caso de usar una función de activación sigmoide utilizaría la función:

$$z = b + \sum_i x_i w_i$$

$$\text{valor de la neurona} = \text{sigmoid}(x) = \frac{1}{1 + e^{-z}}$$

Finalmente, en las redes neuronales cada neurona conectada con otra neurona de otra capa distinta a la suya tendrá un peso los cuales funcionan para amplificar o amortiguar una entrada.

Las redes neuronales son unidireccionales pasando siempre por la capa de entrada → capas ocultas → capa de salida.

### 4.3.1. Clasificadores

Los clasificadores de Deep Learning que se mostrarán a continuación son los utilizados en la parte experimental, ya que existen una gran cantidad de clasificadores de Deep Learning, los cuales se han seleccionado los más utilizados para solucionar este problema, en este caso encontraremos los algoritmos de Convolutional Neural Networks (CNN) y Long Short-Term Memory Units (LSTMs).

#### 4.3.1.1 Convolutional Neural Networks (CNN)

Una CNN se compone de una serie de capas que generar una convolución de los datos de entrada, es decir son filtrados con el objetivo de extraer la máxima información útil. Estas capas tienen una serie de parámetros internos llamados kernel que aprenden. Así los filtros se van ajustando automáticamente sin la necesidad de una selección manual de características (Veličković, 2017), (Araujo dos Santos, L.2016) y (Britz, 2015).

Previamente es necesario explicar la convolución y pooling para entender la transformación realizada para aplicar estas capas a texto.

#### Convolución:

El objetivo de la convolución es extraer las características de una matriz, para ello es utilizado definir una serie de filtros, que modificaran la matriz. Los filtros son matrices más pequeñas  $h \times w$  denominados como "convolution kernel". El cual generara la modificación de la primera. Y este filtro se ira desplazando por la matriz para realizar la multiplicación entre ambas. Este algoritmo se utiliza para imágenes, donde la matriz son los pixeles.

La función utilizada para hacer la convolución entre las dos matrices es:

$$(I * K) = \sum_{i=1}^h \sum_{j=1}^w K_{ij} \cdot I_{x+i-1,y+j-1}$$

Un ejemplo de convolución aplicando la formula anteriores, es el de la Figura 17 que a partir de la matriz de entrada "I" realizando el filtrado a partir de la matriz "K" se obtiene el resultado de "I \* K"

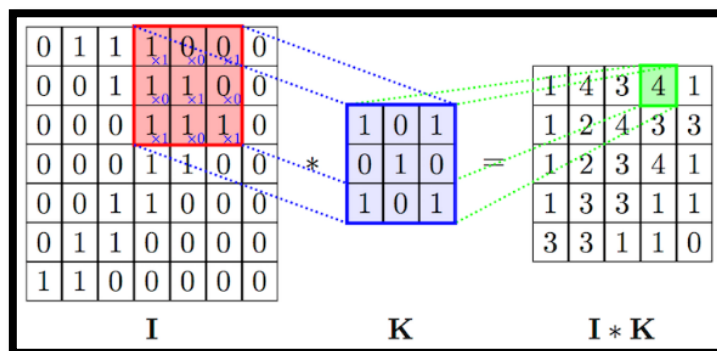


Figura 17 Convolución de una matriz

En la capa de convolución se aplica la formula anterior, pero con las variantes de añadir las bias, la desviación típica y la dimensionalidad d en caso de ser RGB es 3.

$$(I * K) = \sigma(b + \sum_{i=1}^h \sum_{j=1}^w \sum_{k=1}^d K_{ij} \cdot I_{x+i-1,y+j-1})$$

### Pooling

La capa de pooling se basa a partir de una matriz de entrada, esta es transformada a partir de una matriz más pequeña, usualmente suele ser 2 x 2. La cual hace de conceso entre los valores que está matriz puede llegar a agrupar a partir de una función. Se puede apreciar en la Figura 18, como a partir de una función de máximos va agrupando la matriz 4 x 4 en matrices 2 x 2 aplicándoles el máximo a los valores de esta.

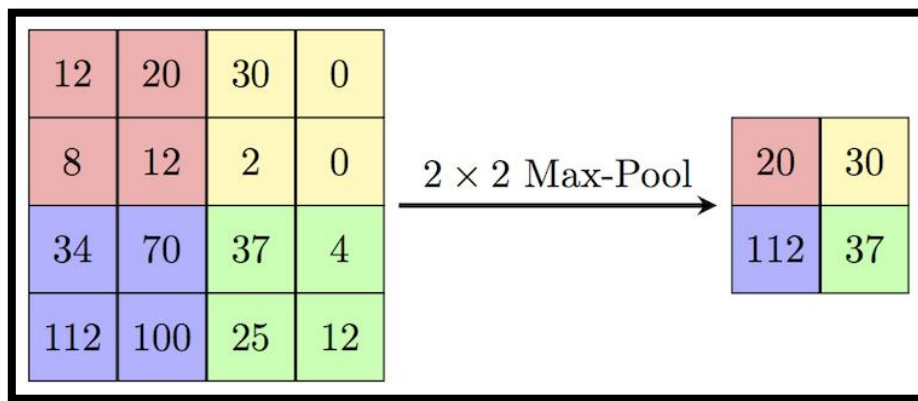


Figura 18 Aplicación matriz pooling

### Aplicación de Capa de Convolución para NLP

Cuando se trata desea aplicar este algoritmo para NLP es necesario adaptar la Matriz, donde cada fila “x” de la matriz corresponde a un token del documento, este token puede ser una palabra o una letra. Las columnas “y” son el vector de representación de la palabra o letra con una dimensionalidad d, estas pueden ser directamente representaciones provenientes de vectores pre entrenados como word2vec o GloVe. O la representación de vectores únicos que representan la palabra en un vocabulario, la matriz generada equivaldría a la imagen en pixeles. Cabe indicar que se generarán N matrices, una por cada documento del corpus. Por ello todos los documentos cuando son transformados a la matriz deben tener la misma longitud. De esta manera, es necesaria la previa trasformación haciendo uso de técnicas de padding como en Tokenización de las palabras.

Los filtros (kernel) aplicados a la matriz, que representa la imagen tiene la peculiaridad que la anchura de esta matriz es la misma que la del vector y la altura es reducida considerablemente al igual que para las imágenes. Por lo que el filtro se ira desplazando por las diferentes filas completas.

Se puede visualizar esta representación en la Figura 19.



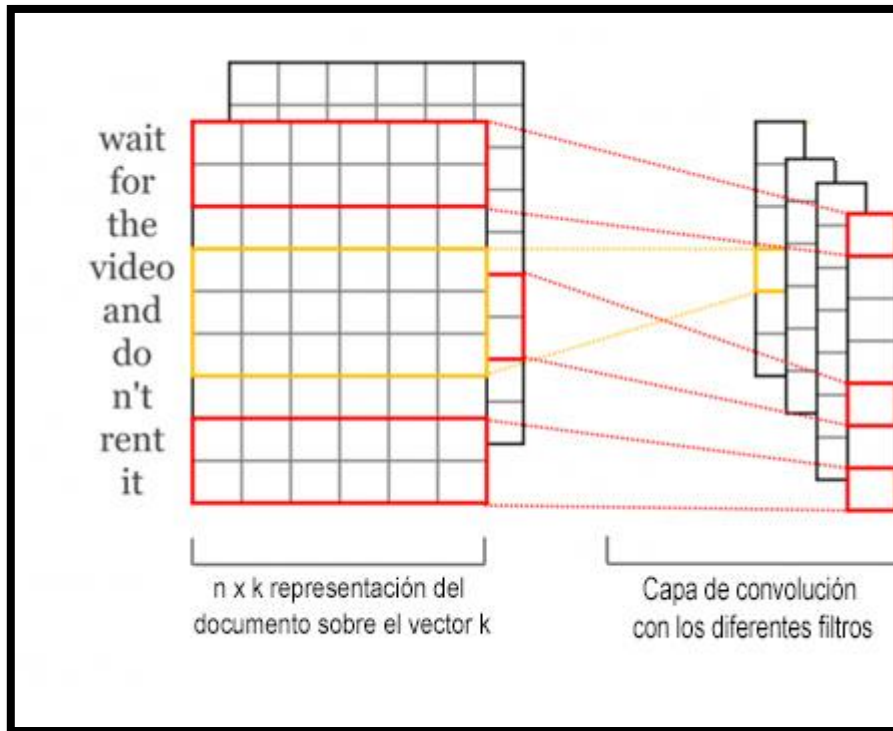


Figura 19 Capa de convolución

### Aplicación pooling

Funciona exactamente igual que para las imágenes, pero con la diferencia que se realiza a partir de cada una de las capas generadas por los resultados de los diferentes filtros aplicados en la capa de convolución.

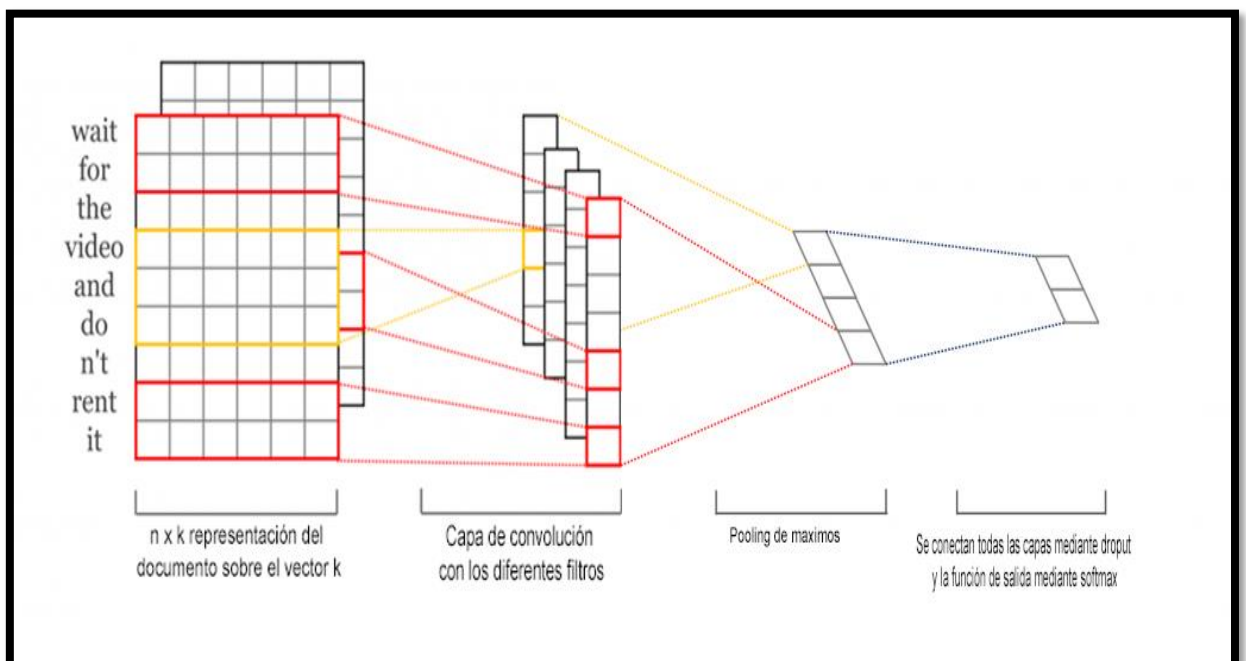
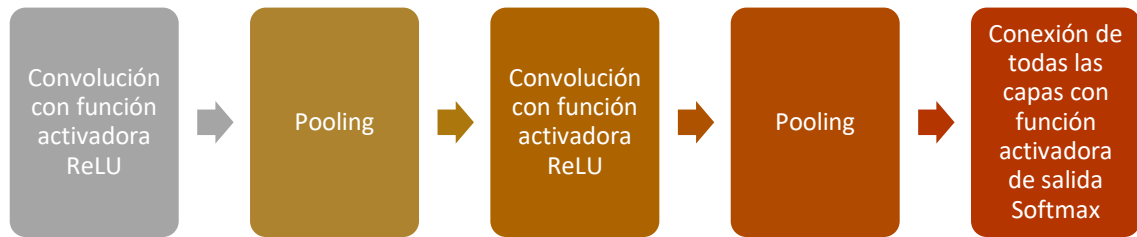


Figura 20 CNN reducida

Para concluir, este algoritmo usualmente en la aplicación de textos está compuesta por las siguientes capas que se muestran a continuación. Para clasificar textos, las cuales han sido implementadas en el proyecto práctico.



#### 4.3.1.2 Recurrent Neural Networks (RNN)

Las redes neuronales recurrentes utilizan las salidas de las neuronas en la capa oculta como una nueva entrada, esto permite que la red neuronal a medida que va recibiendo entradas esta va aprendiendo y va generando un modelo acorde a las nuevas predicciones realizadas por esta. Se puede ver en la Figura 21, como funciona este tipo de algoritmo, donde por cada entrada se genera un nuevo estado de la red neuronal, la cual va actualizando siempre los pesos para ser utilizados posteriormente. Por ello este tipo de red es dependiente de las predicciones realizadas y de la información aprendida por esta. Ya que cada vez que agrega una nueva información transforma la información existente completamente al aplicar la función.

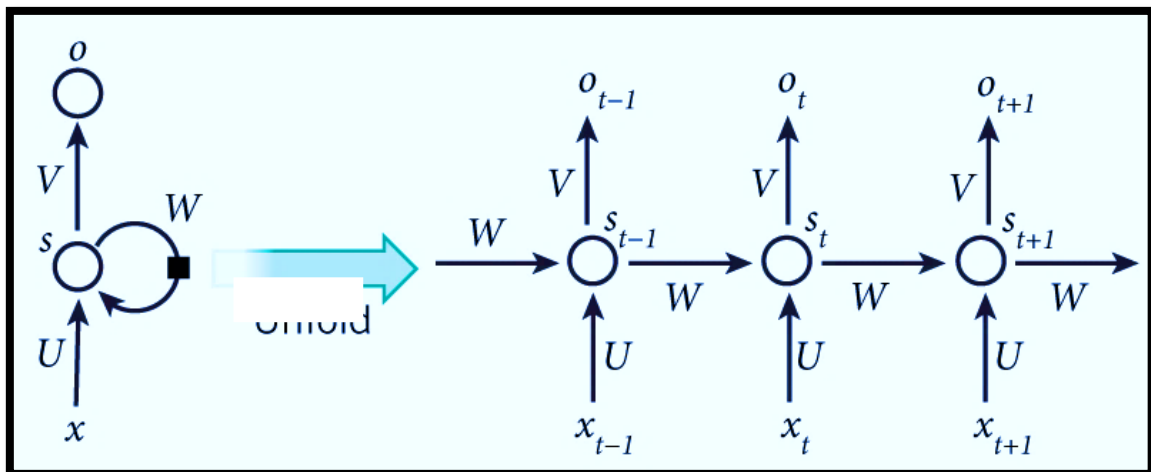


Figura 21 RNN

Las RNN funcionan bien cuando trabaja con dependencias a corto plazo. En cambio, con dependencias a largo plazo estas fallan, ya que no son capaces de entender el contexto que hay por cada entrada. Esto genera, que una predicción realizada muy anteriormente no podrá recordarlo para realizar predicciones en el presente.

### 4.3.1.3 Long Short-Term Memory Units (LSTMs)

Es un algoritmo basado en las RNN, pero en vez de transformar toda la información existente en las capas ocultas, esta se basa en hacer pequeñas modificaciones a la información mediante multiplicaciones y adiciones.

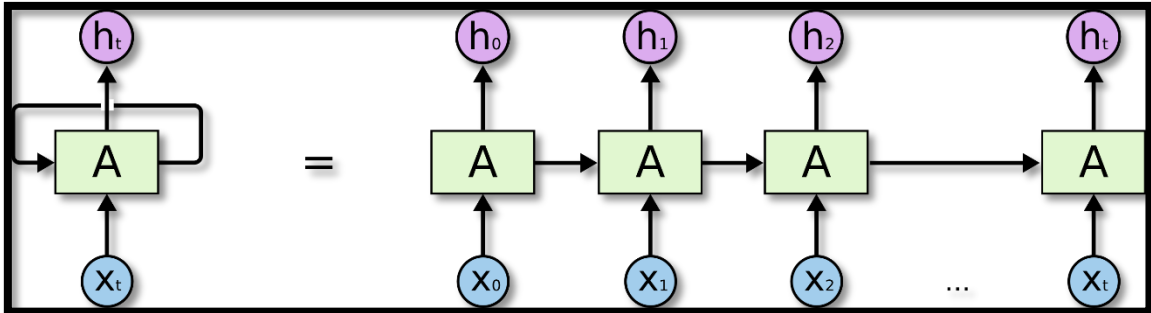


Figura 22 LSTMs

Las LSTM implementan un mecanismo llamado estados celulares. En la cual la información fluye a través de este. De esta manera, el algoritmo LSTM puede recordar u olvidar de manera selectiva.

Cada neurona esta vez puede corresponder a 3 estados distintos. El estado anterior, el cual mantiene la información una información previa. El estado oculto anterior, el cual mantiene el mismo resultado que en la neurona anterior. La entrada en el paso del tiempo actual, la cual obtiene el valor que se está alimentando en ese momento.

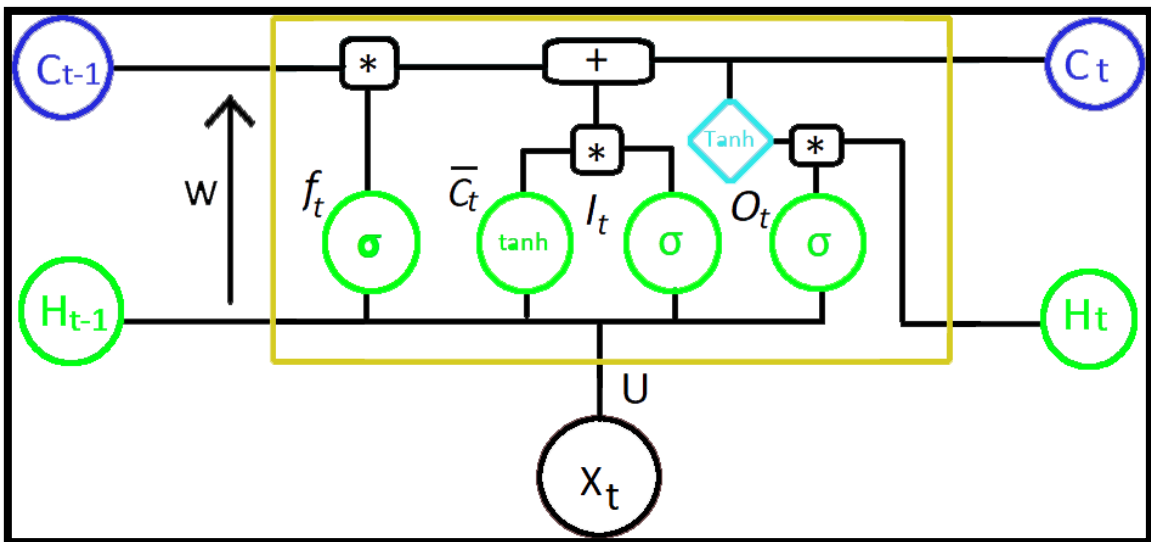


Figura 23 Neurona LSTM

La representación de la Figura 23 junto a sus funciones empleadas son:

$X_t$  = Vector de entrada actual, proveniente de la capa de entrada.

$H_{t-1}$  = Salida de la anterior neurona

$C_{t-1}$  = Valor del estado celular de la neurona (memoria).

$C_t$  = Memoria de la neurona actual.

$H_t$  = Salida de la neurona actual.

$W, U$  = pesos de los diferentes vectores de las neuronas.

\* = multiplicación sabia de los elementos.

+ = adición sabia de los elementos.

$$f_t = \sigma (X_t \cdot U_f + H_{t-1} \cdot W_f)$$

$$\bar{C}_t = \tan (X_t \cdot U_f + H_{t-1} \cdot W_f)$$

$$I_t = \sigma (X_t \cdot U_i + H_{t-1} \cdot W_i)$$

$$O_t = \sigma (X_t \cdot U_o + H_{t-1} \cdot W_o)$$

Por ello el resultado de las salidas es:

$$C_t = f_t \cdot C_{t-1} + I_t \cdot \bar{C}_t$$

$$H_t = O_t \cdot \tan(C_t)$$

## 5.Desarrollo practico

### 5.1. Infraestructura

Antes de comenzar con el desarrollo de la investigación, previamente se tuvo que realizar una investigación para determinar que lenguaje de programación se utilizaría, si se emplearía algún control de versiones, también conocer cuál sería la estructura principal de los archivos, la licencia con la cual se publicaría y finalmente las herramientas de creación (IDE) y visualización de código (Jupyter Notebook).

#### 5.1.1. Lenguaje

Para comenzar con la experimentación y uso de algoritmos para la investigación, ha sido necesaria la realización de comparaciones entre los distintos lenguajes de programación. De entre los lenguajes, Java, C++, R, Python, de entre otros como se puede observar en la Figura 24, el que actualmente da un mejor rendimiento y tiene mucha documentación para generar clasificadores es Python, por ello para toda la investigación ha sido realizada mediante este con la última versión de este, la 3.6.1. Utilizando la configuración de Anaconda.

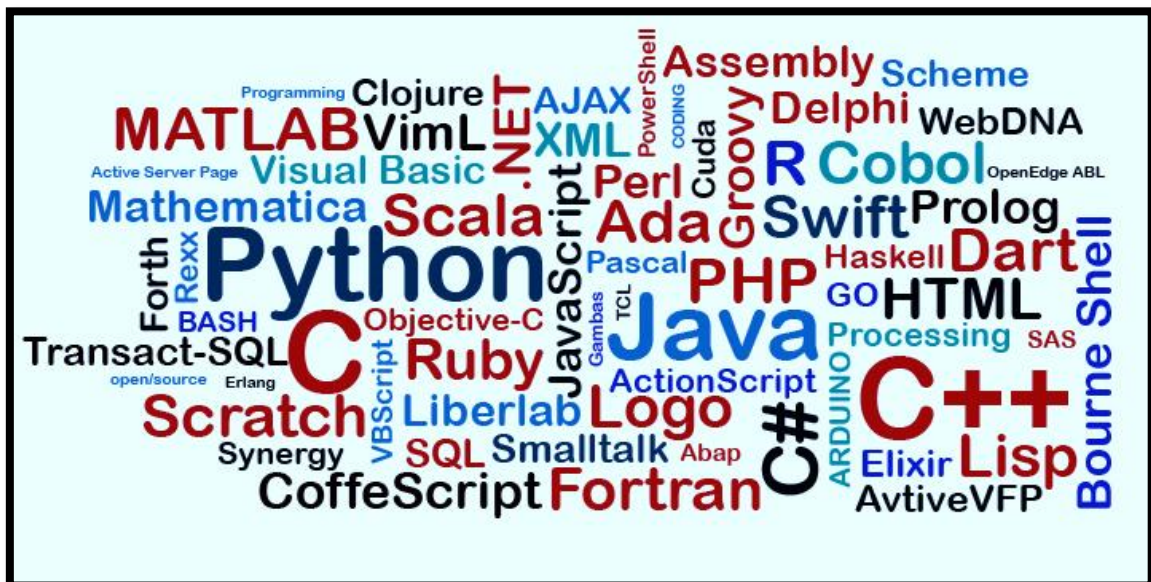


Figura 24 Lenguajes de programación

## 5.1.2. Control de versiones

En este apartado, se tuvo en cuenta el mantener un control de versiones, para así poder tener todos los estados del proyecto, permitiendo que en cualquier momento se pueda volver a un estado anterior en caso de que fuese necesario o se requiera recuperar una función previa utilizada. Así pues, se seleccionó Git utilizando su plataforma libre de publicación del repositorio en Github. Pues este permite publicar y sincronizar todo el proyecto sobre Git. Además, en el caso de que un usuario quiera realizar sus propias pruebas, puede clonarse el proyecto y utilizarlo para su propio uso.

De manera que, en la Figura 25 se puede observar las diferentes actualizaciones que se han publicado en la plataforma Github. Y en la Figura 26 las contribuciones realizadas.

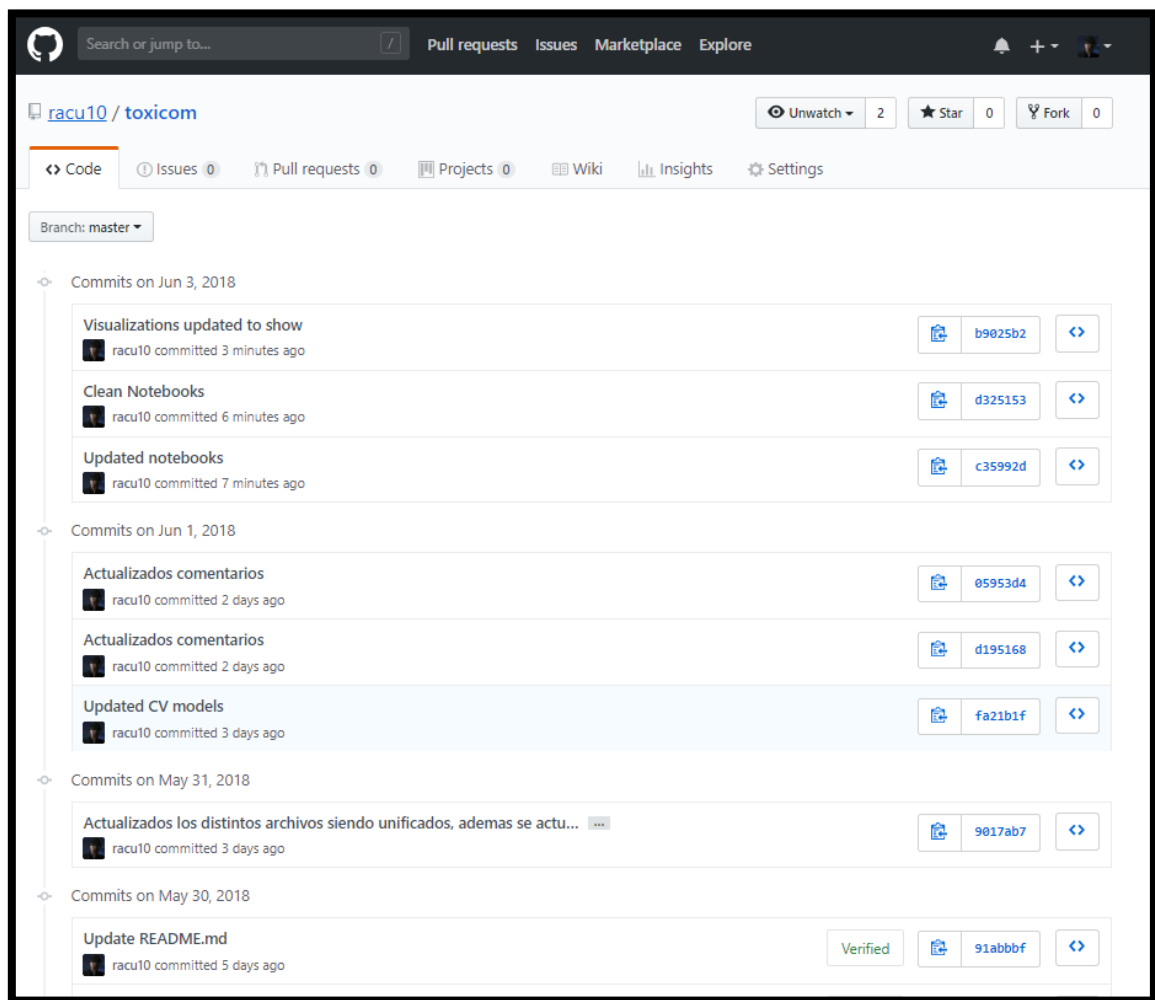


Figura 25 Publicaciones github

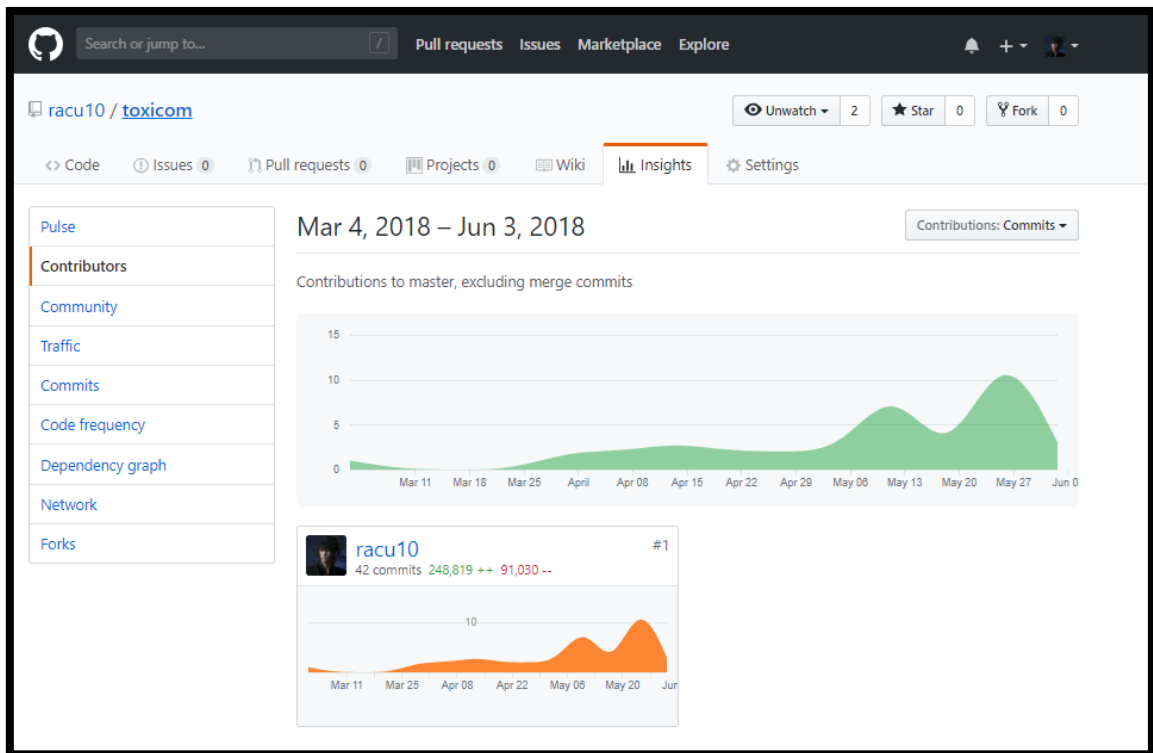


Figura 26 Contribuciones al proyecto



### 5.1.3. Estructura de Archivos

Para la creación del proyecto de investigación, fue imprescindible una previa investigación para conocer el mejor modo de estructurar su sistema de archivos. Puesto que es importante mantener una estructura lógica, dado que esto facilita el tratamiento de los datos de cada apartado del proyecto. Del mismo modo, es necesario conocer que hay en cada carpeta por si se desean realizar futuras modificaciones en el análisis de los datos.

Así pues, por ello se seleccionó Cookicutter, dado que esta librería estudia diferentes estructuras típicas de proyectos. Y de entre sus modelos se seleccionó Cookicutter Data Science, cuya estructura de archivos es la que se usa comúnmente en un proyecto de data science. Su representación se muestra en la Figura 27.



Figura 27 Cookicutter organización del proyecto

Por consiguiente, es necesario mencionar que el motivo por el cual se ha seleccionado esta estructura, es por las múltiples ventajas que aporta. Por un lado, porque dispone de una estructura muy definida indicando la ubicación lógica de todos los archivos. Por otro, porque al utilizar el lenguaje Python genera archivos Tox, permitiendo introducir las librerías de dependencia del proyecto dentro de requirements.txt, y este al ser ejecutado, las instala permitiendo que se pueda utilizar el proyecto directamente.

Y finalmente cabe añadir, que está pensada para proyectos en Git. Pues genera los archivos gitignore y gitkeep. Estos permiten la exclusión de aquellas carpetas que no sean necesarias publicar dentro del repositorio Git. Por otro lado, además genera automáticamente la licencia dentro del proyecto, seleccionando de una lista la que se desea utilizar.

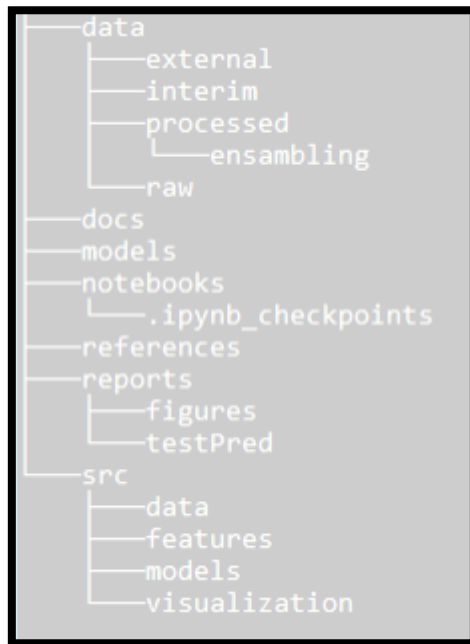


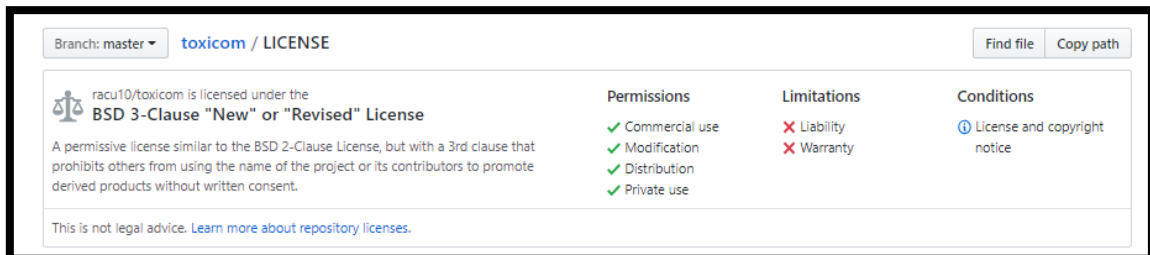
Figura 28 Sistema de archivos final

### 5.1.4. Licencia del proyecto

Como bien se sabe, la licencia es muy importante en un proyecto. Pues si se desea que este sea público y de fácil acceso y no por el contrario privativo es necesario seleccionar la licencia más adecuada. En este caso, previamente se investigaron las más comunes las cuales son MIT y BSD. A continuación, se explican más detalladamente cada una de ellas.

- MIT (Massachusetts Institute of Technology): Se trata de una licencia permisiva y que además no dispone de copyright. Esto permite que el código fuente pueda ser reutilizado y modificado totalmente. Cabe añadir, que normalmente se utiliza en proyectos de software libre.
- BSD (Berkeley Software Distribution): Es una licencia muy similar a MIT puesto que es permisiva ya que, cualquier persona puede utilizarla y modificar aquello que desee. No obstante, en este caso cuando se modifica y por lo tanto se crea una nueva versión, debe de aparecer públicamente. Esto permitirá que la librería pueda ser mejorada continuamente.
- Privativa: Es una licencia cerrada en la que el código fuente únicamente puede ser modificada y leída por su creador. Por lo tanto, los usuarios ni siquiera la pueden utilizar/adaptar para su propio uso.

Así pues, en el caso de la investigación se seleccionó la licencia BSD. Puesto que la intención es que en el futuro el código fuente este constantemente actualizándose para mejorar los sistemas de clasificación de mensajes y, por lo tanto, pueda evolucionar y mejorar de forma progresiva.



The screenshot shows the GitHub interface for the 'toxicom / LICENSE' file. At the top, it indicates the branch is 'master'. The license is identified as 'BSD 3-Clause "New" or "Revised" License'. A brief description states it is a permissive license similar to BSD 2-Clause but with a 3rd clause prohibiting the use of the project name for promotion without consent. A table summarizes the license terms:

Permissions	Limitations	Conditions
<input checked="" type="checkbox"/> Commercial use	<input checked="" type="checkbox"/> Liability	<input checked="" type="checkbox"/> License and copyright notice
<input checked="" type="checkbox"/> Modification	<input checked="" type="checkbox"/> Warranty	
<input checked="" type="checkbox"/> Distribution		
<input checked="" type="checkbox"/> Private use		

At the bottom, there is a disclaimer: 'This is not legal advice. Learn more about repository licenses.'

Figura 29 Licencia del proyecto

### 5.1.5. Visualización y creación del código

Para la visualización del código, se utilizaron los Notebooks de Jupyter, ya que Github permite leerlos correctamente. Además de que muestra el código por apartados. Todo esto, permite tener una amplia visión del código, de este modo las personas que decidan ver el código no tendrán la necesidad de descargarlo y así lo podrán visualizar para su aplicación o analítica a realizar.

Como se puede observar en la Figura 30, desde el propio navegador web se puede visualizar tanto el código como los gráficos generados.

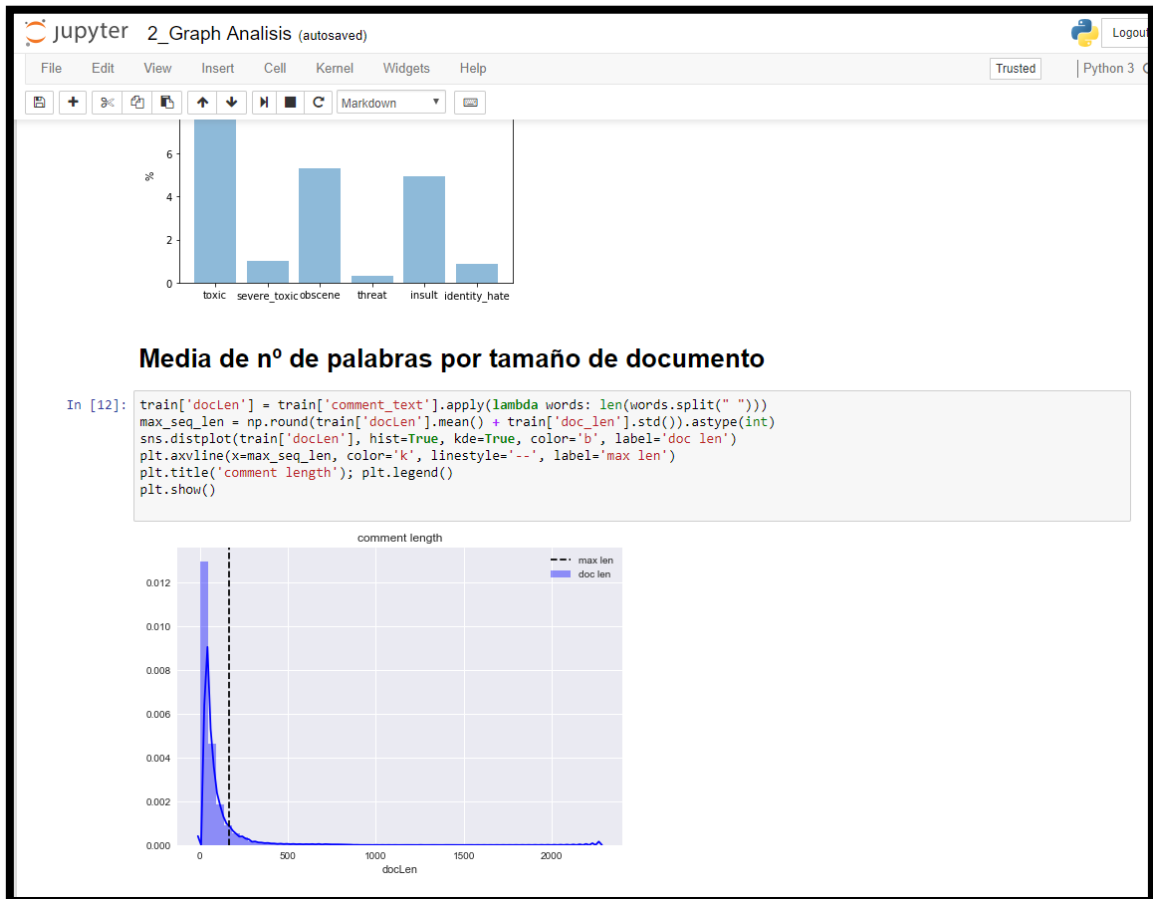


Figura 30 Jupyter Notebook <https://github.com/racu10/toxicom/tree/master/notebooks>

## 5.2. Desarrollo de experimentos

En este apartado se mostrarán cada una de las acciones realizadas para clasificar los comentarios de la Wikipedia. Provenientes de la competición de Kaggle, Toxic Comment Classification Challenge - Identify and classify toxic online comments.

### 5.2.1. Limpieza de los datos

El primer notebook que encontramos es el de clean words. En este apartado todos los comentarios que se disponen tanto de entrenamiento como test, son limpiados. Este proceso se basa en tres partes fundamentales.

En primer lugar, se hace una visión previa de los datos que se disponen. Estos estarán almacenados en la carpeta de data → raw. En este caso la estructura de los datos de entrenamiento la podemos visualizar en la Figura 31. La cual está compuesta para cada identificador, un comentario y en las columnas posteriores es indicada cada clase con un 1 en caso que corresponda con la columna y un 0 en caso contrario. En cambio, en los datos de test se puede observar en Figura 32, como únicamente los datos tienen un identificador y un comentario. Estos datos serán utilizados para predecir y mediante la web de Kaggle, se podrá verificar la precisión obtenida.

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0

Figura 31 Datos de entrenamiento

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.
5	0001ea8717f6de06	Thank you for understanding. I think very high...
6	00024115d4cbde0f	Please do not add nonsense to Wikipedia. Such ...
7	000247e83dcc1211	:Dear god this site is horrible.
8	00025358d4737918	" \n Only a fool can believe in such numbers. ...
9	00026d1092fe71cc	== Double Redirects == \n\n When fixing double...

Figura 32 Datos de test

En segundo lugar, se lematizan todas y cada una de las palabras de los comentarios. Con el objetivo de tener las palabras unificadas, es decir son transformadas en la palabra lema la cual representa cada una de sus variaciones. Tanto para los datos de entrenamiento y test.

En tercer lugar, son eliminadas aquellas palabras que son consideradas Stopwords de los comentarios. Estas son un subconjunto de palabras las cuales no tienen un valor propio para la clasificación de textos, como por ejemplo son los artículos.

En cuarto lugar, con las palabras resultantes de cada comentario se vinculará a la posición a la cual el comentario pertenecía, pero con la peculiaridad que tendrá dos estructuras distintas y por lo tanto será asignado cada tipo a una columna nueva generada. La primera será una lista de las palabras resultantes por comentario, esta columna es denominada listOfCleanWords. La segunda será la unión de cada una de las palabras como si fuese un texto simple, esta columna es denominada cleanWordsAsText.

La estructura final, se puede ver en la Figura 33 para los datos de entrenamiento y en la Figura 34 para los datos de test. Para facilitar las lecturas necesarias sin la necesidad de realizar constantemente este proceso sobre los datos.

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	listOfCleanWords	cleanWordsAsText
0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	['explanation', 'edits', 'made', 'username', '...']	explanation edits made username hardcore metal...
000103f0d9c9cb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	['aww', 'match', 'background', 'colour', 'seem...']	aww match background colour seemingly stuck th...
000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	['hey', 'man', 'really', 'trying', 'edit', 'wa...']	hey man really trying edit war guy constantly ...
0001b41b1c6bb37e	"\nMore!\nI can't make any real suggestions on ...	0	0	0	0	0	0	['make', 'real', 'suggestion', 'improvement', '...']	make real suggestion improvement wondered sect...
0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	['sir', 'hero', 'chance', 'remember', 'page']	sir hero chance remember page

Figura 33 Datos de entrenamiento con las nuevas columnas añadidas

id	comment_text	listOfCleanWords	cleanWordsAsText
00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...	['yo', 'bitch', 'ja', 'rule', 'succesful', 'ev...']	yo bitch ja rule succesful ever whats hat sad ...
0000247867823ef7	== From RfC == \n\n The title is fine as it is...	['rfc', 'title', 'fine', 'imo']	rfc title fine imo
00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	['source', 'zawe', 'ashton', 'lapland']	source zawe ashton lapland
00017563c3f7919a	:if you have a look back at the source, the in...	['look', 'back', 'source', 'information', 'upd...']	look back source information update correct fo...
00017695ad8997eb	I don't anonymously edit articles at all.	['anonymously', 'edit', 'article']	anonymously edit article

Figura 34 Datos de test con las nuevas columnas añadidas

Finalmente, las estructuras nuevas son almacenadas en la carpeta de data → processed.

## 5.2.2. Visualización de los datos de entrenamiento

Este apartado va dirigido al segundo Notebook generado, llamado Graph Analysis. En este Notebook se mostrarán diferentes visualizaciones realizadas sobre los datos, por tal de entender mejor como su distribución.

### 5.2.2.1 Porcentaje de los comentarios por clase sobre el total de comentarios.

En la Figura 35, se muestra el porcentaje de comentarios que se tienen en el conjunto de datos de entrenamiento, por cada clase. La cual permite observar que el mayor conjunto de datos clasificados es Tóxico, aunque cabe decir que tener un porcentaje inferior a 10, indica que no se disponen de una gran cantidad de muestras.

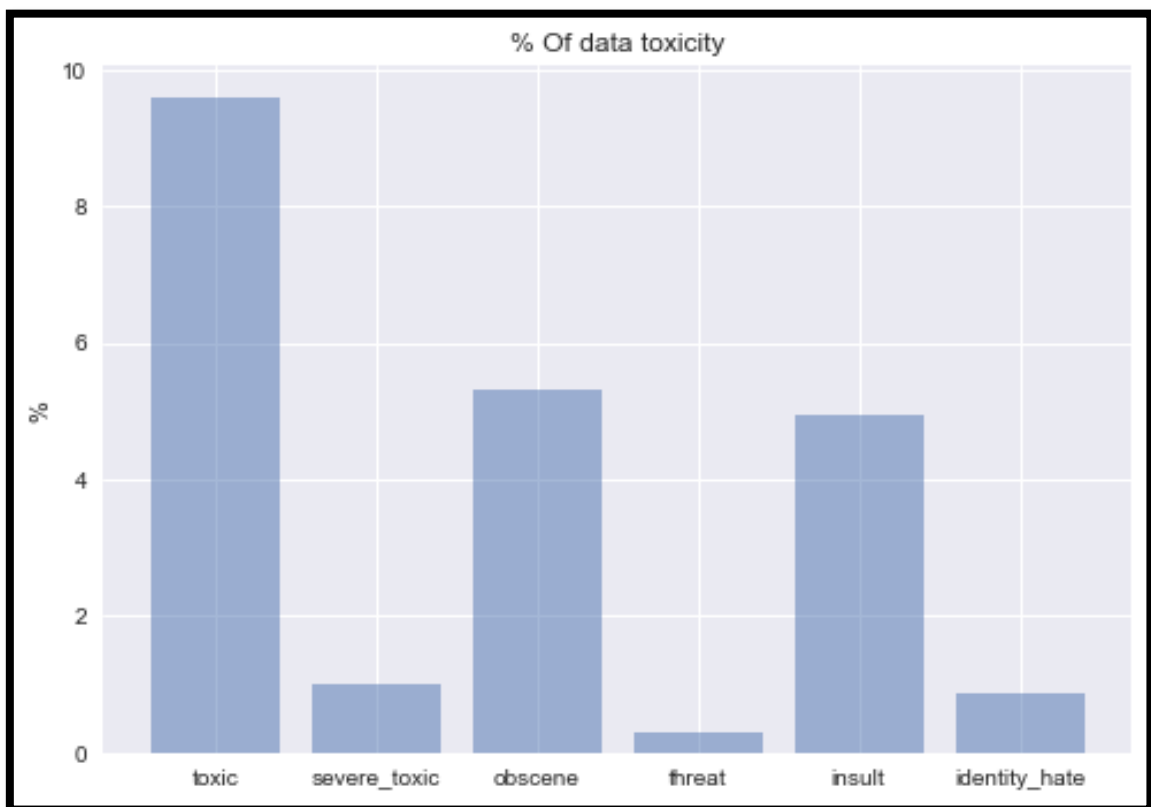


Figura 35 % Of data toxicity



### 5.2.2.2 Cantidad de comentarios por clase incluyendo los comentarios limpios

Con la Figura 36 se puede hacer una idea más exacta de la diversidad de tipos de comentarios que se tienen clasificados. Cabe mencionar que la proporción de comentarios limpios es muy grande en comparación con el resto.

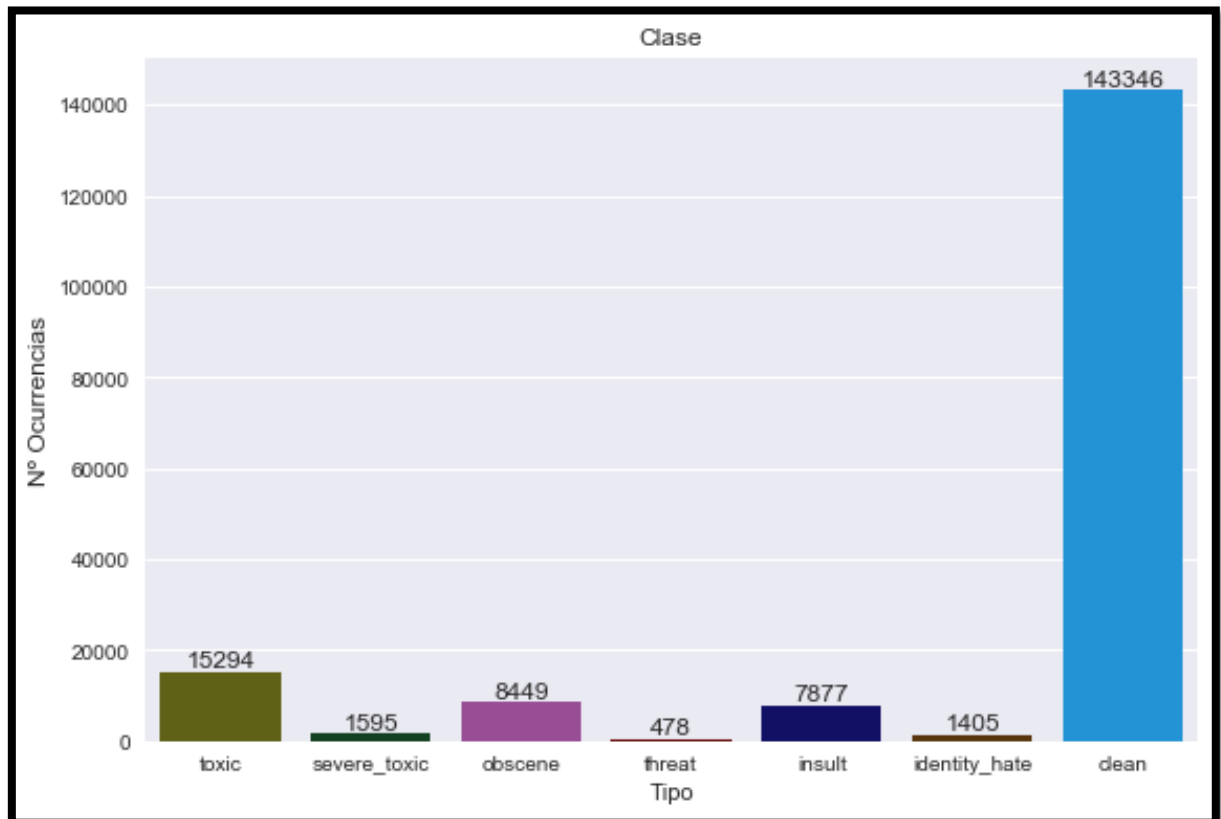


Figura 36 N.º de comentarios por tipo



### 5.2.2.4 Palabras más repetidas por categorías

Con esta serie de figuras se puede llegar a ver realmente que palabras y con qué cantidad son las más repetidas dentro de una categoría. Con la finalidad de obtener una idea principal de las palabras que posiblemente afecten bastante a cada categoría. Además de comprobar si hay palabras que afectan en más de una. Como es el caso de “Fuck” que se utiliza muchísimo en las categorías Toxic, Severe Toxic y obscene.

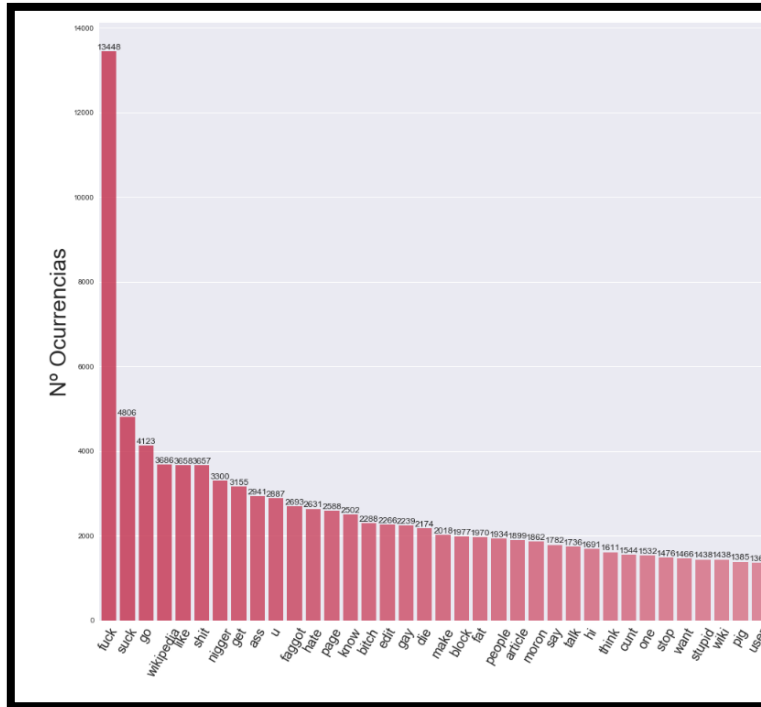


Figura 43 Palabras más repetidas Toxic

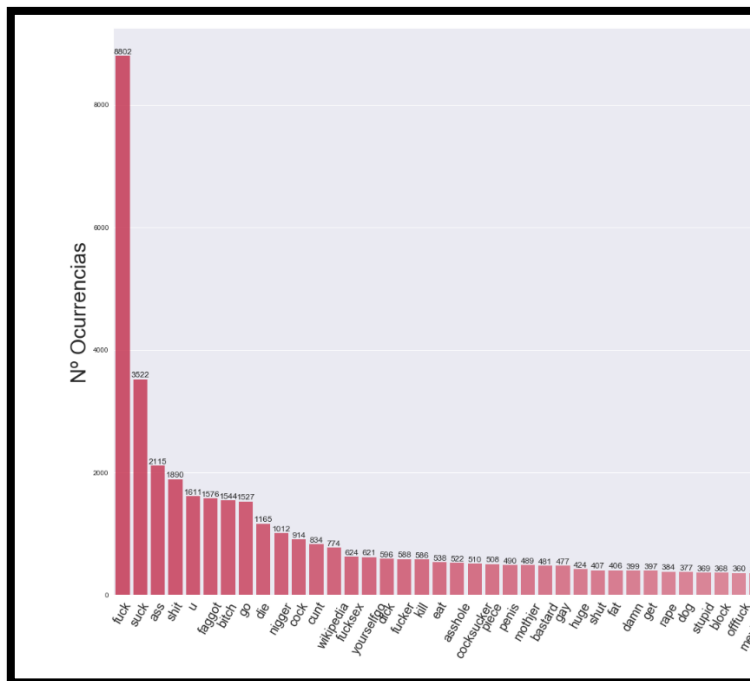


Figura 44 Palabras más repetidas Severe Toxic

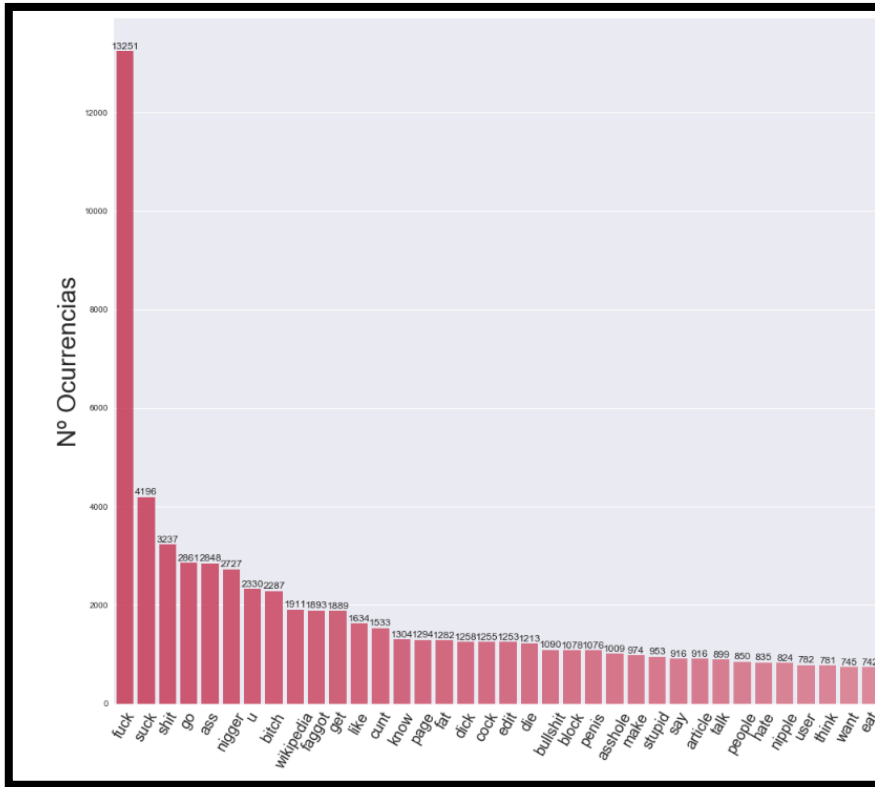


Figura 45 Palabras más repetidas Obscene

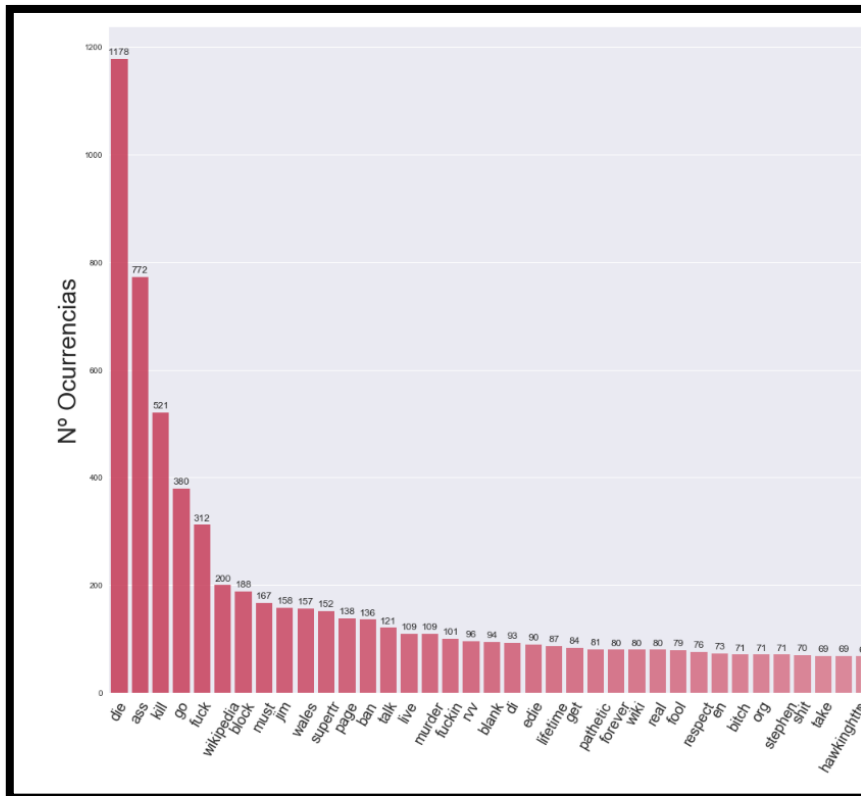


Figura 46 Palabras más repetidas Threat

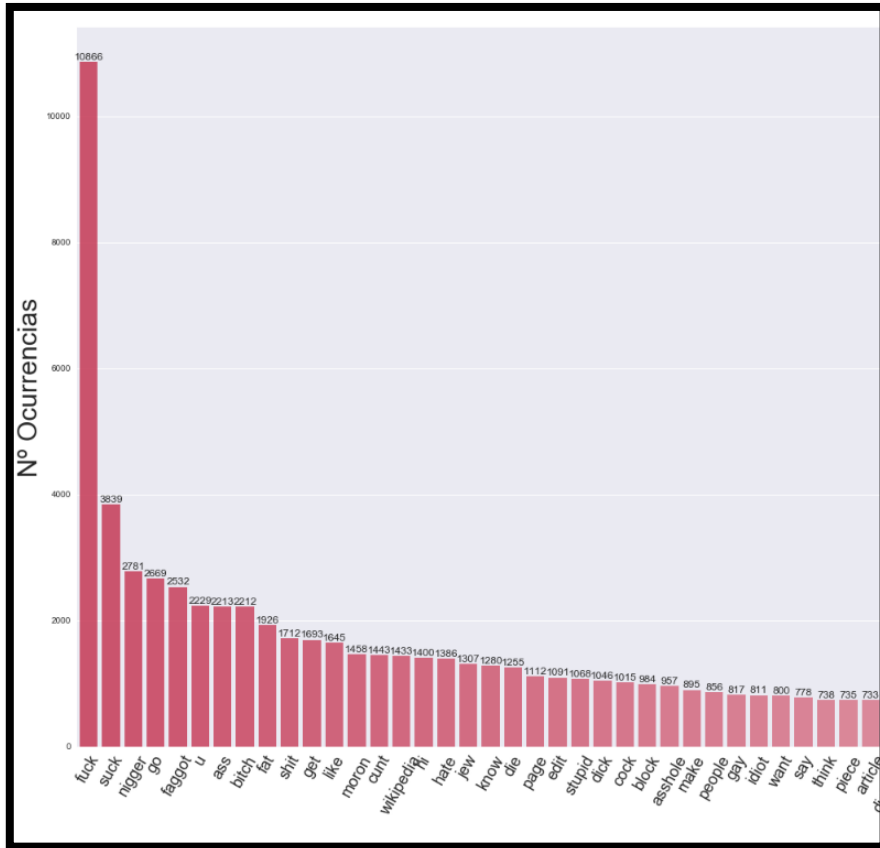


Figura 47 Palabras más repetidas Insult

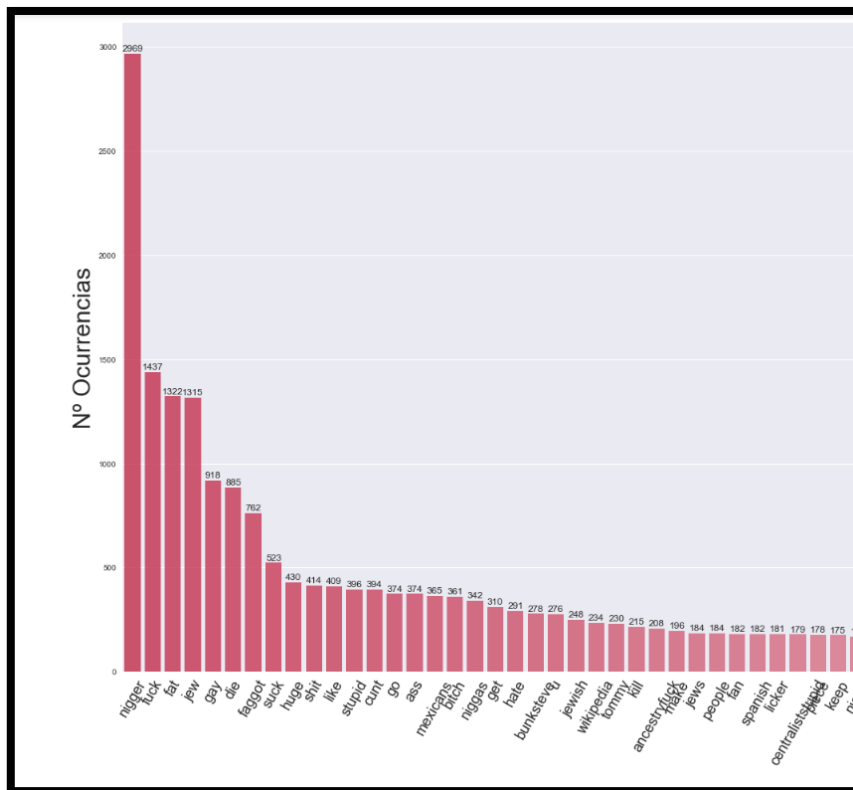


Figura 48 Palabras más repetidas Identity Hate

### 5.2.3. Clasificación por Machine Learning y Deep Learning

Este apartado corresponde al archivo "Análisis of clean words\_Machine\_Learning", el cual contiene todos los algoritmos implementados de Machine Learning y "Análisis of clean words\_Deep\_Learning" para los algoritmos implementados de Deep Learning.

Cabe mencionar, que los datos utilizados son los de la competición de Kaggle "Toxic Comment Classification Challenge", dados por el proyecto Jigsaw de Google. La posibilidad de medir el porcentaje de precisión es únicamente evaluando test y subiéndolo a la web de Kaggle. El máximo de subidas diarias es de 5.

En primer lugar, se procedió a la realización de entrenamientos basados en las mejores características. Es decir, se seleccionaron las Top N características de los comentarios. Con la peculiaridad que previamente eran separados los comentarios según si eran o no tóxicos y por lo tanto aproximadamente el 50% de características pertenecía a un tipo y el otro 50% al otro, aun así, cabe decir que no era exactamente 50% debido que las palabras seleccionadas podían llegar a coincidir. La estructura de los datos utilizada fue la de TF-IDF. Dado que daba mejores resultados que el resto. Al realizar este proceso mediante Cross Validation, tal y como se puede apreciar en la Figura 49. A excepción del algoritmo Decision Tree, que al principio tenía un buen porcentaje luego bajaba y luego volvía a incrementar debido al overfitting, el resto de algoritmos incrementaban constantemente hasta que se estabilizaban en aproximadamente 0.915% de precisión.

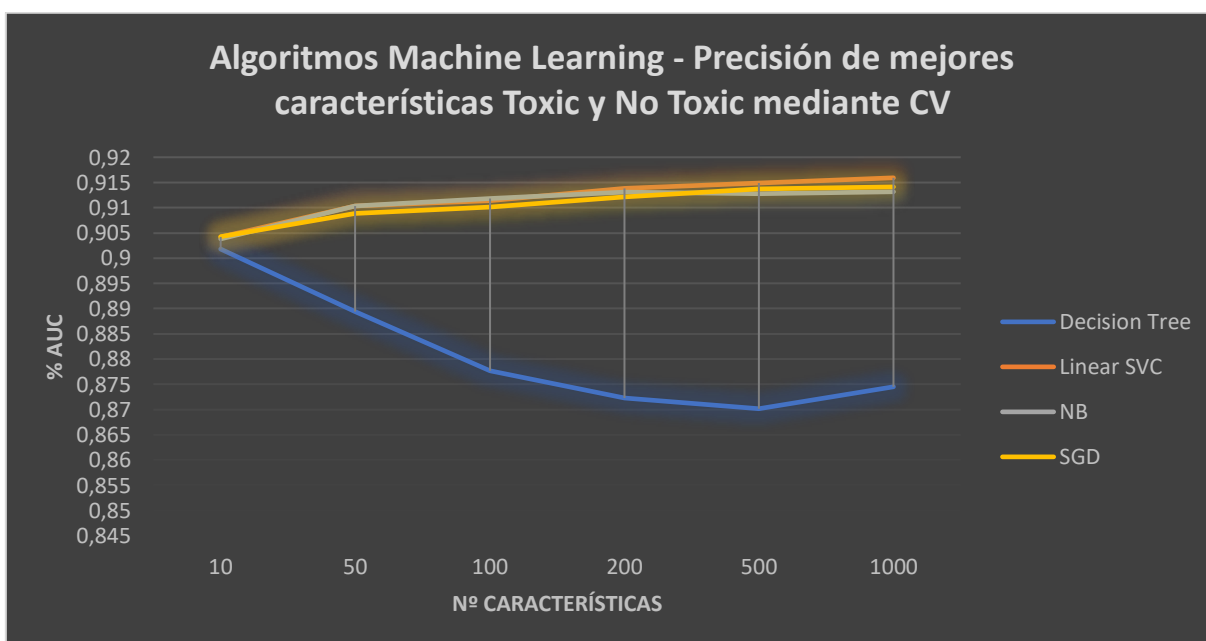


Figura 49 Algoritmos Machine Learning sobre CV de Train basado 50% toxic 50% clean

Posteriormente se procedió a la predicción de test, con los mejores resultados obtenidos por los algoritmos previamente. Tal y como se muestra en la Figura 50, se puede apreciar que los resultados obtenidos como máximo se llegaba a una precisión aproximada del 70%.

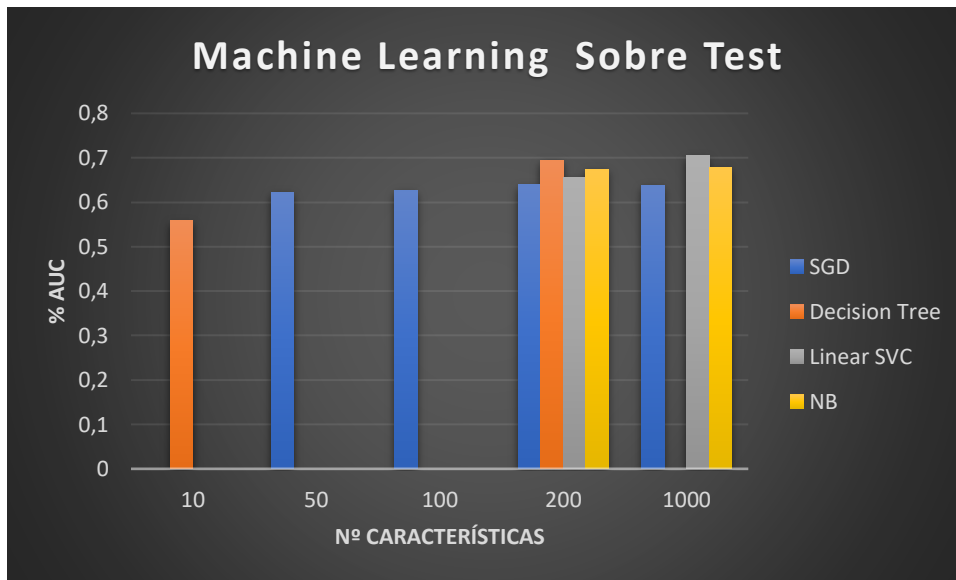


Figura 50 Algoritmos ML sobre test basado 50% toxic 50% clean

Siguiendo con la situación anterior se procedió con los algoritmos de Deep Learning. Tal y como se puede apreciar en la Figura 51 ambos algoritmos aprendían, y se estancaban sobre 91,8% cuando se realizaban CV sobre el entrenamiento.

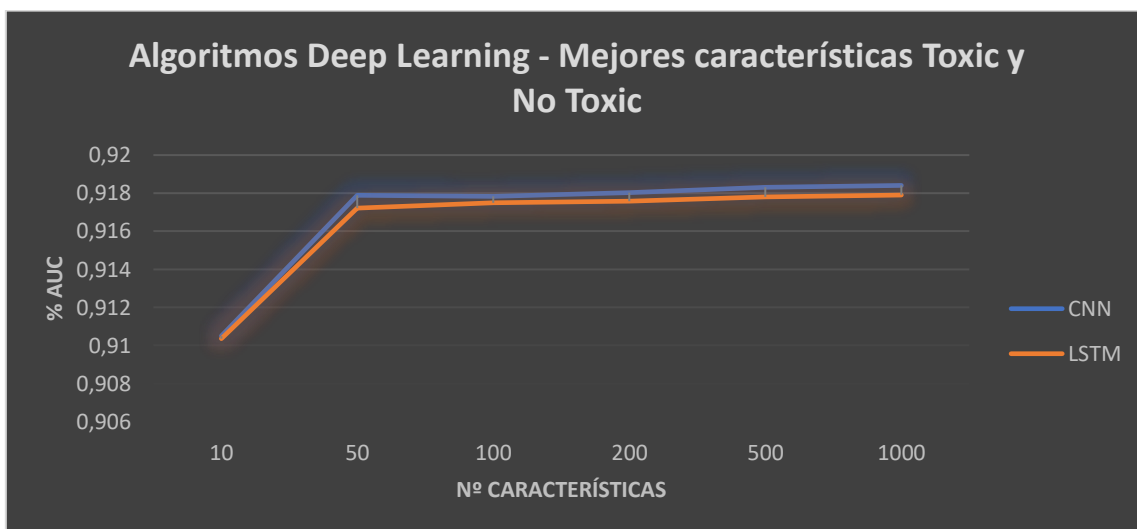


Figura 51 Algoritmos DL sobre CV de Train basado 50% toxic 50% clean

Posteriormente se procedió tal y como paso con Machine Learning a la realización sobre test. En la Figura 52, se puede apreciar que los resultados obtenidos eran bastante parecidos que los de Machine Learning. También la precisión rondaba sobre el 70%.

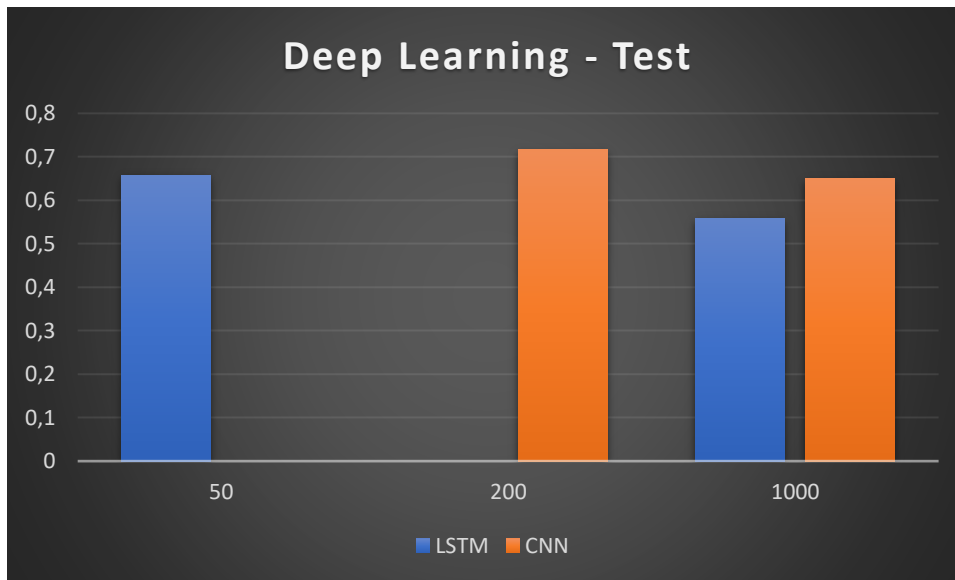


Figura 52 Algoritmos Deep Learning sobre test basado 50% toxic 50% clean

Comparando los resultados obtenidos, con los de los competidores de Kaggle eran bastante pésimos. Así que se procedió a realizar algunos cambios.

En primer lugar, en vez de tratar los subconjuntos de datos de manera dividida para obtener las mejores características se procedió directamente mediante TF-IDF a calcular sobre todo el conjunto. Esto permite que el algoritmo sea mucho más rápido ya que únicamente se debe de calcular el TF-IDF de cada característica. Permitiendo así obtener características superiores a 100000. Eliminar la limitación de las mejores N características obtenidas, hace que los algoritmos a la hora de clasificar tengan un mayor resultado.

En segundo lugar, se procedió a implementar el algoritmo Logistic Regression (Regresión Logística), tras la realización de cómo mejorar los porcentajes, se hayo que este algoritmo daba mejores resultados que el resto. Así pues, se añadió para comprobar si realmente tenía mejores resultados.

En tercer lugar, para cambiar los algoritmos de Deep Learning se procedieron a la implementación de dos nuevas estructuras Tokenización de palabras y Word2vec. Aunque tras realizar la primera prueba con Word2vec sobre test, con el máximo de numero de características que anteriormente se probó un total de 1000 este dio como resultado del 49.74%. Así pues, se decidió que fuese eliminado.

Dado estos cambios, se procedió a la comparativa entre los diferentes algoritmos de Machine Learning utilizando como estructuras, TF-IDF y Matriz termino documento para realizar una predicción sobre test. El número de características fue de 158627. Este valor es el que todos los algoritmos de Machine Learning eran capaces de obtener unos valores de precisión óptimos. Se puede apreciar en la Figura 53, que con el algoritmo TF-IDF se puede llegar a conseguir mejores resultados que con la Matriz TD. Aunque, con ciertos algoritmos con la Matriz TD mejoran un poco respecto a TF-IDF, como es el caso de NB. Otra característica a observar. Es que el algoritmo de Logistic Regression es el que mejor resultado de precisión tiene con un 97.36%.



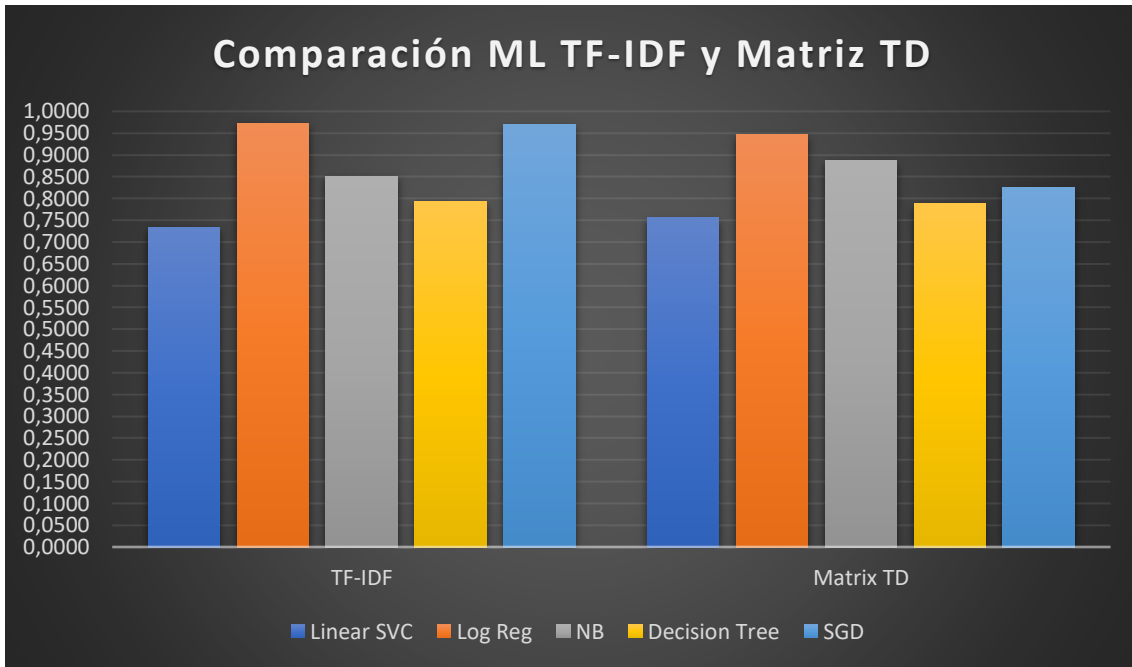


Figura 53 Comparación ML estructuras TF-IDF y Matriz TD con 158627 características

A continuación, en la Figura 54 se muestra los resultados sobre test realizados con Deep Learning y la estructura de Tokenización de las palabras. Se puede apreciar que con 50000 características el algoritmo CNN una precisión de 96.0%. Y el algoritmo LSTM con 50000 características tiene una precisión del 95.83%. Así pues, el algoritmo que da mejores resultados de Deep Learning es el CNN, aunque cabe destacar que ambos están bastante igualados, a excepción de 158627 características que LSTM llega a perder muchísima más precisión que CNN.

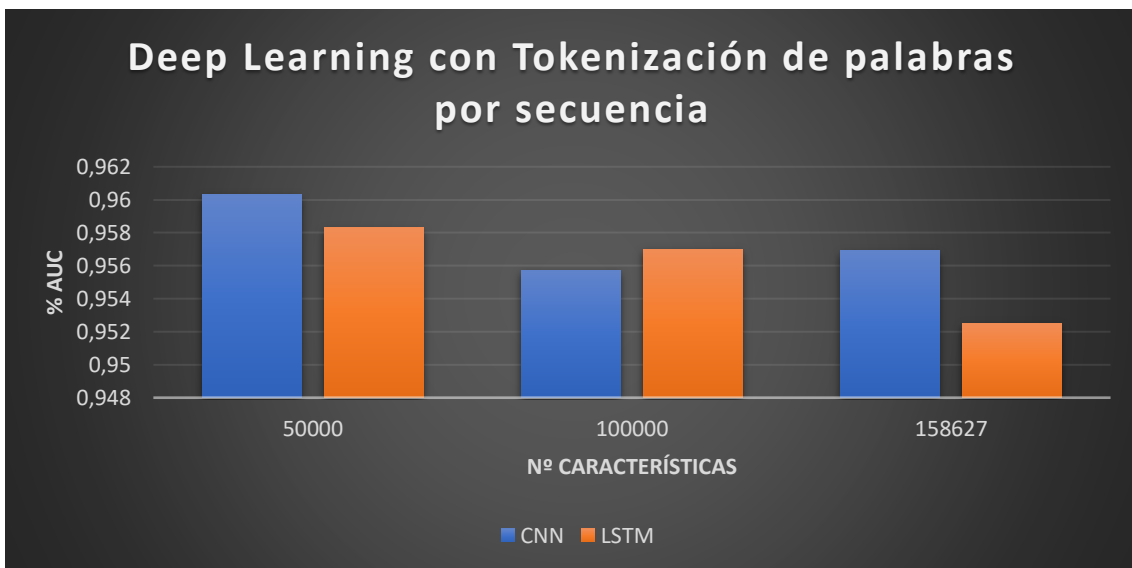


Figura 54 Deep Learning con Tokenización de palabras por secuencia

El siguiente paso a realizar fue intentar optimizar el porcentaje de acierto de la Logistic Regression, frente a test. Para ello, gracias al uso del TF-IDF no solo se utilizaron las palabras individuales, sino que también se utilizaron la combinación de estas. Realizando todas las combinaciones posibles entre 1 y 6 palabras. Una vez obtenido el resultado 158627 características se procedió a aumentar este valor al máximo de combinaciones posibles del corpus. El segundo paso fue realizar los dos mismos experimentos anteriores pero seleccionado las combinaciones entre 2 y 6 palabras. Y finalmente bajar el número de características a 100000 y a 50000 con las combinaciones de 1 a 6 palabras. Tal y como se puede apreciar en la Figura 55, si aumentamos el número de características a un número muy elevado, el clasificador da igual si tiene las combinaciones entre 1 y 6 palabras o 2 y 6 no es capaz de predecir correctamente y su precisión baja a 0,5. En cambio, si nos fijamos con 100000 características predecir con bigramas a hexagramas sale también 0,5. Debido a que los unigramas son quienes aportan un mayor peso para clasificar textos. De ese modo, apoyándose en los unigramas y teniendo la posibilidad de predecir con algún N-grama de soporte, es capaz de diferenciar mejor entre el tipo de mensajes que únicamente con unigramas. Es por eso que la precisión con 100000 es de 0,9736 superando los 0,9721 con menos características, por lo que el aprendizaje será más rápido.

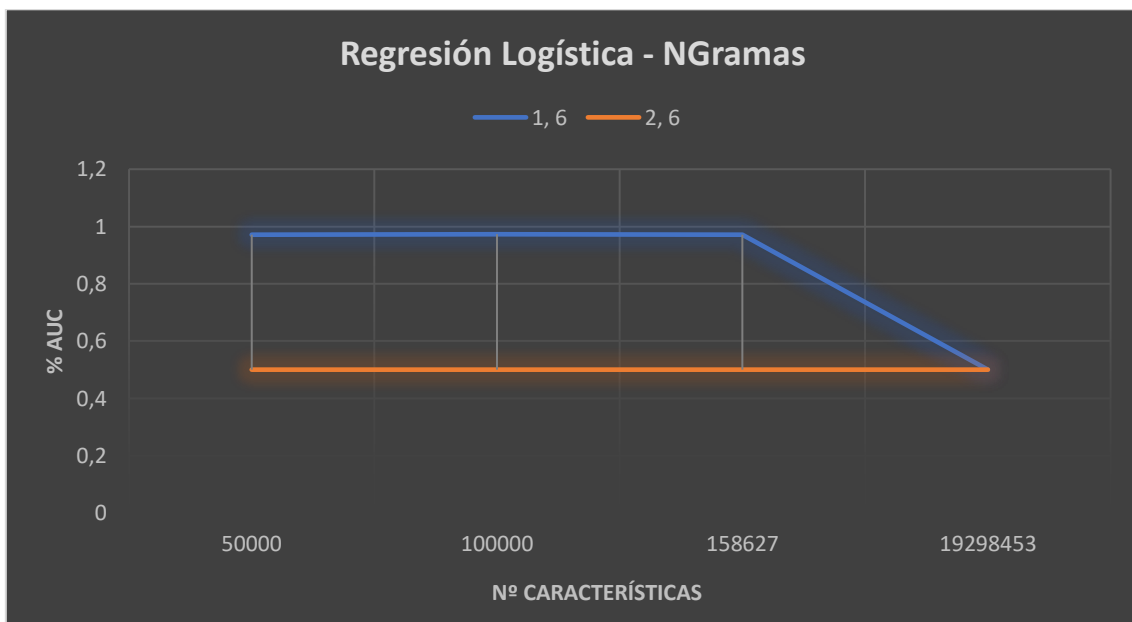


Figura 55 Regresión Logística - NGramas

## 5.2.4. Ensembling

Este apartado corresponde al archivo “Ensembling Test Submissions”, cuyo objetivo es mediante la clasificación realizada por N clasificadores, a cada clasificación realizada se le asigna un peso, mediante esta asignación se espera mejorar la precisión de los clasificadores. Esto se basa en hacer una especie de consenso entre los clasificadores por tal de potenciar la salida.

Otra forma de hacer ensembling es por la correlación de las clasificaciones previas, realizadas por los algoritmos vistos previamente, los cuales han generado sus respectivas predicciones almacenadas en ficheros. Para la realización de este ensembling se basa cinco pasos diferentes.

En primer lugar, se busca la correlación entre parejas de archivos, donde cada archivo tiene la predicción realizada.

En segundo lugar, se unen los archivos con mayor correlación en un único archivo, a partir de las predicciones medias.

En tercer lugar, se elimina de la lista que gestiona que archivos han sido unidos.

En cuarto lugar, se repite los pasos 2 y 3 hasta que queda el último archivo, debido a que todas las predicciones han sido unidas.

Finalmente, en la Figura 56 se puede apreciar los diferentes ensemblings, de los algoritmos utilizados previamente. Seleccionando aquellas predicciones que han sido la mejor por cada uno. Aquellos que tiene un valor significa que se ha utilizado la proporcionalidad de esta y los que no tienen valores se ha realizado mediante la media de clases utilizadas. A excepción de “Ensembling Correlacion”, que se ha utilizado mediante la correlación de todos los algoritmos utilizados con la mejor de sus precisiones.

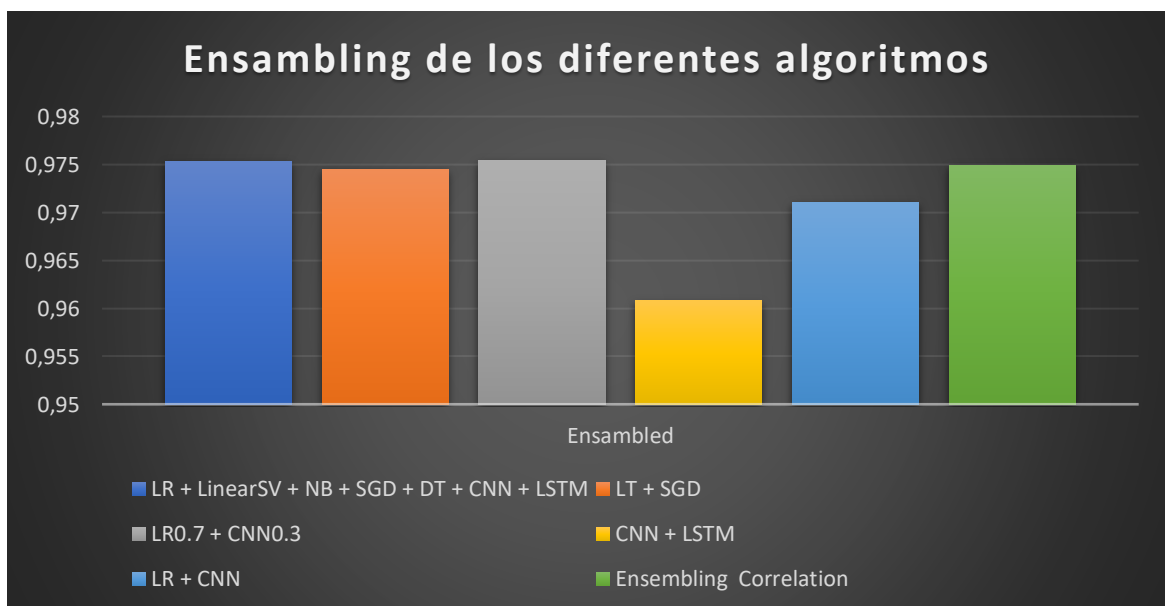


Figura 56 Ensembling de los diferentes algoritmos

### **5.2.5. Parámetros utilizados en el desarrollo de los experimentos**

En cuanto al uso de estructura de textos, los parámetros utilizados para los experimentos de TF-IDF han sido como N-Gramas de 1 a 6 debido, a que para clasificar los resultados eran mejores que con únicamente unigramas. También se han realizado pruebas con únicamente bigramas pero no eran capaces de clasificar correctamente los algoritmos.

En cuanto a las Cross Validation se han ejecutado mediante un fold de 5 y Random. Para realizar un orden aleatorio en los datos tanto de entrenamiento como test.

En cuanto a los algoritmos de regresión logística y Linear SVC se ha utilizado una C de 5.

En el algoritmo de optimización SGD la función de pérdida se ha generado mediante una "log".

Para los árboles de decisión, por tal de no tener un sobre entrenamiento se ha acabado utilizando una profundidad máxima de 8.

En cuanto a los algoritmos de Deep Learning, para la CNN la dimensión utilizada del vector es de 100, las convoluciones han sido realizadas con un total de 128 filtros, con activación relu. La regularización del kernel ha sido mediante un peso de  $1e-4$  con un total de 8 épocas de entrenamiento. También ha sido utilizado un optimizador Adam con  $lr=0.001$ ,  $beta_1=0.9$ ,  $beta_2=0.999$ ,  $epsilon=1e-08$ ,  $decay=0.0$ . Para el modelo LSTM se ha utilizado un vector de 128 como embedding. Con un total de 90 células. Y 60 unidades para para la activación relu de dense. Y como optimizador el mismo que para la CNN. El total de épocas utilizado ha sido 2 por el tiempo que llega a tarda en aprender.

Para todas las salidas de los algoritmos de Deep Learning se ha utilizado una activación sigmoid para 7 clases.

En cuanto a los valores utilizados en ensembling han sido detallados en la Tabla 6.

## 6. Resultados

En primer lugar, en la Tabla 1 se muestran los datos obtenidos por la clasificación Cross Validation de los diferentes algoritmos, con la que se puede hacer una idea que haciendo uso de una CNN con 1000 características se podría llegar a tener una buena precisión AUC ya que esta dando un total del 91,79%, para la realización de estas pruebas eran realizadas mediante la estructura de palabras por TF-IDF.

Algoritmo	N.º Características					
	10	50	100	200	500	1000
Decision Tree	0,901799	0,889435	0,87771	0,872245	0,870158	0,874514
Linear SVC	0,904193	0,910297	0,911444	0,9138	0,91484	0,915925
NB	0,903811	0,910303	0,911814	0,913054	0,912785	0,913161
SGD	0,904262	0,908824	0,910084	0,912177	0,913744	0,914126
CNN	0,910496	0,917883	0,917831	0,918025	0,91832	0,918405
LSTM	0,910353	0,917205	0,91749	0,917568	0,917804	0,917901

Tabla 1 Cross Validation de mejores características 50% toxic 50% clean

Posteriormente se procedió a la realización de obtener los resultados tal y como se muestra en la Tabla 2, sobre el conjunto de test. Para obtener la precisión en AUC es necesario publicarlo en la web de Kaggle, donde está incorporado el proyecto. Cabe indicar, como se ha mencionado anteriormente que Kaggle tiene un límite de subidas diario de clasificaciones realizadas. Por ello se empezaron con aquellos clasificadores que habían dado mejores resultados, en decreciente.

Algoritmo	N.º Características				
	10	50	100	200	1000
Decision Tree				0,6941	
Linear SVC	0,5585	0,6221		0,6561	0,7049
NB				0,6747	0,6784
SGD			0,6275	0,6390	0,6373
CNN				0,7161	0,6492
CNN + W2V					0,4974
LSTM		0,6573			0,5582

Tabla 2 Predicción sobre test de mejores características 50% toxic 50% clean

En cuanto se llevaban unos días subiendo publicaciones y los resultados no eran lo suficientemente buenos. Se dejaron de subir pruebas a Kaggle y se procedió a la investigación de nuevos algoritmos y configuraciones distintas. Tanto de Machine Learning como de Deep Learning. Así pues, se empezaron a realizar pruebas, la mayor inconveniencia de la realización de pruebas es el tiempo de aprendizaje de los algoritmos de Deep Learning. Capaces de llegar a estar consumiendo la CPU al 100% de un I7 de séptima generación, y llegando en según cuales pruebas hasta un uso de +10GB de RAM, tardando +10 horas de ejecución si las pruebas no fallaban a la mitad.

Una vez se encontraron mejores formas de mejorar los algoritmos utilizados se pusieron en práctica, en primer lugar, permitiendo la incrementación del N.º de características, pero siendo optimizadas en memoria, permitiendo que los algoritmos puedan llegar a aprender de más características. A la vez que, con la investigación anterior, se pudo llegar a concretar que el algoritmo de Machine Learning Regresión Logística, para el conjunto de datos obtenido daba muy buenos resultados. Otro aspecto a tener en cuenta era el hacer uso de la Matriz TD en los algoritmos de Machine Learning. Por otro lado, se halló que la tokenización de palabras con padding daba muy buenos resultados.

Así se procedió a la implementación de dichas modificaciones en primer lugar se muestran los cambios realizados en Machine Learning en la Tabla 3. Realizando una comparación entre TF-IDF y Matriz TD, añadiendo el algoritmo de Regresión Logística y con el máximo de palabras que el algoritmo recomendaba como top palabras, en total son 158627. Realizando todas las predicciones sobre test.

Algoritmo	TF-IDF	Matriz TD
Linear SVC	0,7330	0,7553
Log Reg	0,9729	0,9465
NB	0,8515	0,8865
Decision Tree	0,7930	0,7876
SGD	0,9689	0,8243

Tabla 3 Comparación entre TF-IDF y Matriz TD con 158627 características sobre test

Esta vez los resultados se habían incrementado considerablemente, en comparación con los realizados previamente. teniendo como mejor algoritmo la Regresión Logística con un AUC del 97,29%, seguidamente de SGD con 96,89%. Teniendo ya en este caso unos porcentajes muy aceptables. Seguidamente tal y como se muestra en la Tabla 4. Se procedió a la realización de la implementación sobre los algoritmos de Deep Learning contra test. En este caso se utilizó la estructura de palabras con tokenización estableciendo como máximo N características. Y las capas de ambos fueron modificadas.

Algoritmo	N.º Características		
	50000	100000	158627
CNN	0,9603	0,9557	0,9569
LSTM	0,9583	0,9570	0,9525

Tabla 4 Algoritmos Deep Learning predicciones sobre Test

En este caso, el resultado fue muy decente, aunque los entrenamientos son excesivamente pesados de ejecutar. Posteriormente, se procedió a la mejora del mejor clasificador obtenido, en este caso la Regresión Logística. Con ello se encontró que la mejor forma de mejorar un algoritmo de Machine Learning es tener unas mejores características. Así pues, se realizó el entrenamiento ya no solo con unigramas que es como se trabaja previamente, sino también con bigramas en adelante. Además, de realizar unas pruebas si el algoritmo era capaz de aprender correctamente sin unigramas. Se procedió seleccionando un total entre 1 y 6 gramas. Empezando con el valor máximo de combinaciones, luego el recomendado previamente,

posteriormente con 10000. Tal y como se muestra en la Tabla 5. El algoritmo con solo bigramas en adelante no es capaz de aprender correctamente, a la vez que si se tienen demasiadas combinaciones tampoco. Posteriormente el valor con 158627 características el valor no aumentaba respecto a los unigramas, se procedió con 100000 características, y en este caso sí que incremento el AUC. Dado que al bajar el N.º de características este incremento, se procedió una vez más con 50000. Y ya en este caso volvió a decrecer.

N-Gramas	Nº Características			
	50000	100000	158627	19298453
1 - 6	0,9722	0,9736	0,9721	0,5
2 - 6	0,5	0,5	0,5	0,5

Tabla 5 Logistic Regression modificando los N-gramas utilizados y el N.º Características sobre test

Finalmente, se investigó como mejorar el AUC obtenido y mejorar los resultados, en ese momento se empezó a implementar Ensembling de los datos previos, realizados contra test. Tal y como se muestra en la Tabla 6, realizando este algoritmo, los valores de AUC mejoraron considerablemente. Cambiando los pesos y manteniendo siempre que el mayor peso recaiga sobre el algoritmo que mejor ha funcionado individualmente o que todos sobre los que se estén evaluando tengan el mismo peso. Dando en una gran parte de las pruebas realizadas valores de AUC superiores al 97%. Y mejorando la precisión máxima obtenida hasta el momento a un 97,54%.

Peso a cada clasificador							Roc ACC	
LR	LSVC	NB	SGD	DT	CNN	LSTM	Comentario	
0,16	0,14	0,14	0,14	0,14	0,14	0,14	Todos iguales	0,9753
0,5			0,5				Top ML	0,9745
0,7						0,3	Mejor ML + Mejor DL con porcentajes según su AUC	0,9754
					0,5	0,5	Solo DL	0,9608
0,5					0,5		Mejor ML + Mejor DL	0,971
							Ensembling por correlación	0,9749

Tabla 6 Ensembling sobre las predicciones de Test Top de cada algoritmo

## 7. Conclusiones

En este proyecto se han tratado, técnicas para el procesamiento de los textos en lenguaje natural, sobre comentarios de la Wikipedia en Ingles. Haciendo uso de los algoritmos de Machine Learning: Regresión Logística, Naive Bayes, Maquinas de soporte vectorial con un kernel lineal, Arboles de decisión y optimización de máquinas de soporte vectorial con Stochastic Gradient Descent. Y los algoritmos de Deep Learning: Convolutional Neural Network y Long Short-Term Memory Units. Con el objetivo de clasificar los documentos en Toxic, Severe Toxic, Obscene, Threat, Insult, Identity Hate y Neutral.

Dado la realización de varios test, el modelo más robusto es mediante el ensembling, con el cual se ha llegado a tener una precisión de AUC del 97,54%. Esto se debe a que, gracias a la unión de los diferentes clasificadores, son capaces de suplir lo que otro clasificador no ha sido capaz de reconocer una clase, cuando ambos indican que están seguros sobre una clase tiene una alta probabilidad de que así sea y cuando hay diferencias según los porcentajes dados a esa clase es capaz de mejorar si pertenece o no a la clase indicada. Otra cosa a tener en cuenta el algoritmo de ensembling es muy rápido, llegando a dar resultados en pocos minutos, según la cantidad de algoritmos utilizados. La desventaja de dicho algoritmo es que necesita ser alimentado por los demás algoritmos, esto influye en que si un algoritmo que tiene un gran peso da una mala predicción es muy condicionante a que los resultados sean mejores que al haber hecho uso de un solo algoritmo. O si una gran parte de las predicciones que se han realizado han sido erróneas, y en cambio algún algoritmo ha sido capaz de identificar correctamente la clase, igualmente es posible que no mejore la precisión.

Como trabajos a futuro, sería la posibilidad de implementar cada uno de estos algoritmos en otros idiomas diferentes, para la clasificación automática del odio de los diferentes mensajes en los medios sociales que son escritos en diferentes idiomas. Es el caso por ejemplo de que, en España, que se menciona por el periódico que Twitter y Facebook borran menos mensajes de odio. Por otro lado, sería interesante la aplicación de clasificadores de odio en los videojuegos en línea. Con el reto de que sean capaces de detectar y predecir que un mensaje es de odio en el idioma detectado, en tiempo "real", para poder ser evaluado si es mostrado o no.



# Bibliografía

Araujo dos Santos, L. (7 de 12 de 2015). *Neural networks*. Obtenido de leonardoaraujosantos: [https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/neural\\_networks.html](https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/neural_networks.html)

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017). *Deep learning for hate speech detection in tweets*. *26th ACM WWW Companion*, 759–760.dav.

Bernard Shaw, G. (17 de 8 de 2017). *El odio es la venganza de un cobarde intimidado*. Obtenido de Reasons Why: <https://www.reasonwhy.es/reportaje/como-podemos-frenar-el-odio-en-redes-sociales>

Britz, D. (7 de 9 de 2015). *Understanding Convolutional Neural Networks for NLP*. Obtenido de wildml: <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

Brownlee, J. (1 de 4 de 2016). *Logistic Regression for Machine Learning*. Obtenido de machinelearningmastery: <https://machinelearningmastery.com/logistic-regression-for-machine-learning/>

Brownlee, J. (23 de 3 de 2016). *Gradient Descent For Machine Learning*. Obtenido de machinelearningmastery: <https://machinelearningmastery.com/gradient-descent-for-machine-learning/>

Cano, C. (29 de 4 de 2013). *Definición de Web 2.0 y su evolución hacia Web 3.0* Web 2.0 y su evolución hacia Web 3.0. Obtenido de comenzandodecero: <https://comenzandodecero.com/definicion-de-web-2-0/>

Carmona Suárez, E. J. (14 de 7 de 2014). *Tutorial sobre Máquinas de Vectores Soporte (SVM)*. Obtenido de uned: [http://www.ia.uned.es/~ejcarmona/publicaciones/\[2013-Carmona\]%20SVM.pdf](http://www.ia.uned.es/~ejcarmona/publicaciones/[2013-Carmona]%20SVM.pdf)

Chatzakou, D., Kourtellis, N., Blackburn, J., De Cristofaro, E., Stringhini, G., y Vakali, A. (2017). Mean birds: Detecting aggression and bullying on twitter. *9th ACM WebScience*.

Christian. (27 de 3 de 2007). *Amor, odio (Apuntes sobre la filosofía de Empédocles)*. Obtenido de Filosofía La Guía: <https://filosofia.laguia2000.com/filosofia-griega/amor-odio-apuntes-sobre-la-filosofia-de-empedocles>

Cimpanu, C. (17 de 2 de 2017). *Wikipedia Comments Destroyed by a Few Highly Toxic Users*. Obtenido de bleepingcomputer: <https://www.bleepingcomputer.com/news/security/wikipedia-comments-destroyed-by-a-few-highly-toxic-users/>

Clarke, I., y Grieve, J. (2017). Dimensions of abusive language on twitter. *Proceedings of the First Workshop on Abusive Language Online*, 1–10.

Cóndor, R. (2018). Probando mejoras al sistema de bans y reportes. Leagueoflegends. Recuperado de: <https://las.leagueoflegends.com/es/news/game-updates/player-behavior/probando-mejoras-al-sistema-de-bans-y-reportes>

- Davidson, T., Warmsley, D., Macy, M., y Weber, I. (2017). Automated hate speech detection and the problema of offensive language. *arXiv preprint arXiv:1703.04009*.
- Dinakar, K.; Reichart, R., y Lieberman, H. (2011). Modeling the detection of textual cyberbullying. *The Social Mobile Web* 11(02).
- Djuric, N., Zhou, J., Morris, R., Grbovic, M., Radosavljevic, V., y Bhamidipati, N. (2015). Hate speech detection with comment embeddings. *24th ACM WWW*, 29–30.
- Epp, R. (16 de 3 de 2018). Lazy Ensembling Algorithm. págs. <https://www.kaggle.com/reppic/lazy-ensembling-algorithm>.
- Founta, A. M., Chatzakou, D., Kourtellis, N., Blackburn, J., Vakali, A., y Leontiadis, I. (2018). A Unified Deep Learning Architecture for Abuse Detection. *arXiv preprint arXiv:1802.00385*.
- Georgakopoulos, S. V., Tasoulis, S. K., Vrahatis, A. G., y Plagianakos, V. P. (2018). Convolutional Neural Networks for Toxic Comment Classification. *arXiv preprint arXiv:1802.09957*.
- Goldberg, Y. (2016). Neural Network Methods in Natural Language Processing. *Bar Ilan University*
- González, A. (30 de 7 de 2014). *Conceptos básicos de Machine Learning*. Obtenido de CleverData: <http://cleverdata.io/conceptos-basicos-machine-learning/>
- Goodfellow, I., Bengio, Y., Courville, A., y Bengio, Y. (2016). Deep learning (Vol. 1). *Cambridge: MIT press*.
- Hariani, y Riadi, I. (2017). Detection of cyberbullying on social media using data mining techniques. *International Journal of Computer Science and Information Security*.
- Jané, C. (26 de 12 de 2017). *Twitter y Facebook borran menos mensajes de odio en España*. Obtenido de El periodico: <https://www.elperiodico.com/es/sociedad/20171226/espana-borrado-contenidos-odio-twitter-facebook-6515386>
- Kumara, A., Sethib, A., Akhtara, M., Ekbala, A., Biemannc, C., & Bhattacharyyaa, P. (2017). Sentiment Prediction in Financial Text. *SemEval*, 894-904. Obtenido de <https://www.aclweb.org/anthology/S/S17/S17-2.pdf>
- López Fernández, R. (17 de 3 de 2013). *Marketing Digital desde 0*. Obtenido de marketingdigitaldesdecero: <https://marketingdigitaldesdecero.com/2013/03/17/diferencias-entre-medio-social-y-red-social/>
- López, D., Vera, N., & Pedraza, L. (12 de 2016). Analysis of Multilayer Neural Network Modeling. *International Journal of Mathematical and Computational Sciences*, pág. Vol 10. Obtenido de waset: <https://waset.org/publications/10006216/analysis-of-multilayer-neural-network-modeling-and-long-short-term-memory>

- Mayoral, R. (16 de 4 de 2012). *El odio hace nuestra mente más peligrosa de lo que se creía*. Obtenido de elconfidencial: [https://blogs.elconfidencial.com/alma-corazon-vida/divan-digital/2012-04-16/el-odio-hace-nuestra-mente-mas-peligrosa-de-lo-que-se-creia\\_587927/](https://blogs.elconfidencial.com/alma-corazon-vida/divan-digital/2012-04-16/el-odio-hace-nuestra-mente-mas-peligrosa-de-lo-que-se-creia_587927/)
- Mehdad, Y., y Tetreault, J. R. (2016). Do characters abuse more than words?. *SIGDIAL*, 299–303.
- Mülle, M. (28 de 2 de 2018). Naive Bayes Classification With Sklearn. págs. <https://blog.sicara.com/naive-bayes-classifier-sklearn-python-example-tips-42d100429e44>
- NSS. (4 de 6 de 2017). *An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec*. Obtenido de analyticsvidhya: <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec>
- Ratkiewicz, J., Conover, M., Meiss, M. R., Gonçalves, B., Flammini, A., y Menczer, F. (2011). Detecting and tracking political abuse in social media. *ICWSM*, 11, 297-304.
- Ruiz, A. L. (2017). 98 estadísticas de las redes sociales para 2017. Brandwatch. Recuperado de: <https://www.brandwatch.com/es/blog/98-estadisticas-de-las-redes-sociales-para-2017/>
- Shalev-Shwartz, S., y Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. *Cambridge university press*.
- Smith, K. (7 de 6 de 2016). *44 estadísticas de Twitter para 2016 MARKETING*. Obtenido de brandwatch: <https://www.brandwatch.com/es/blog/44-estadisticas-twitter-2016/>
- Veličković, P. (20 de 3 de 2017). Deep learning for complete beginners: convolutional neural networks with keras. *cambridgespark*, págs. <https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html>
- Warner, W., y Hirschberg, J. (2012). Detecting hate speech on the world wide web. *2nd Workshop on Language in Social Media*, 19–26.
- Waseem, Z., y Hovy, D. (2016). Hateful symbols or hateful people? predictive features for hate speech detection on twitter. *SRW@ HLT-NAACL*, 88–93.
- Wulczyn, E., Thain, N., y Dixon, L. (2017). Ex machina: Personal attacks seen at scale. *Proceedings of the 26th International Conference on World Wide Web* (pp. 1391-1399). International World Wide Web Conferences Steering Committee.
- Xiang, G., Fan, B., Wang, L., Hong, J., y Rose, C. (2012). Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. *21st ACM CIKM*, 1980–1984.
- Young, T., Hazarika, D., Poria, S., y Cambria, E. (2018). Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*.

# Anexos

Resultados obtenidos en la plataforma de Kaggle con la puntuación privada y pública del AUC.

Submission and Description	Private Score	Public Score	Use for Final Score
<a href="#">ensambledTopMRDL.csv</a> 21 days ago by Raul <a href="#">add submission details</a>	0.9763	0.9754	<input type="checkbox"/>
<a href="#">ensambledSame.csv</a> 21 days ago by Raul <a href="#">add submission details</a>	0.9752	0.9753	<input type="checkbox"/>
<a href="#">ensambledByCorrelations.csv</a> 2 days ago by Raul <a href="#">add submission details</a>	0.9747	0.9749	<input type="checkbox"/>
<a href="#">ensambledTop2.csv</a> 21 days ago by Raul <a href="#">add submission details</a>	0.9745	0.9745	<input type="checkbox"/>
<a href="#">lr.csv</a> 21 days ago by Raul <a href="#">add submission details</a>	0.9737	0.9736	<input type="checkbox"/>
<a href="#">submission.csv</a> a month ago by Raul <a href="#">add submission details</a>	0.9735	0.9734	<input type="checkbox"/>
<a href="#">LR_TFID100000.csv</a> a month ago by Raul <a href="#">add submission details</a>	0.9730	0.9729	<input type="checkbox"/>
<a href="#">LR_TFID100000.csv</a> a month ago by Raul <a href="#">add submission details</a>	0.9730	0.9729	<input type="checkbox"/>