



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Detección del discurso de odio en Twitter

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Alejandro Pérez Fernández

Tutor: Lluís Felip Hurtado Oliver

Cotutor: Ferran Pla Santamaria

Curso 2019-2020

Resumen

En este Trabajo de Fin de Grado se presenta el diseño e implementación realizado al crear un sistema de alerta que avise de tweets que contengan discurso de odio en su publicación. Para ello, se emplearán diversos tipos de modelos, utilizando técnicas relacionadas con el Aprendizaje Automático, para hacer una clasificación automática de dichas publicaciones. Estos modelos se explicarán y compararán para escoger el que obtenga mejores resultados sobre tweets reales ya etiquetados.

Palabras clave: Detección, Discurso de odio, Aprendizaje Automático, Twitter, Bot, Sistema de Alerta, Python

Resum

En aquest Treball de Fi de Grau, es presenta el disseny i implementació realitzat al crear un sistema d'alerta que avise de tweets que continguen discurs d'odi en la seua publicació. Per a això, s'empraran diversos tipus de models, utilitzant tècniques relacionades amb el Aprenentatge Automàtic, per a fer una classificació automàtica d'aquestes publicacions. Aquests models s'explicaran i compararan per a triar el que obtinga millors resultats sobre tweets reals ja etiquetats.

Paraules clau: Detecció, Discurs d'odi, Aprenentatge Automàtic, Twitter, Sistema d'Alerta, Python

Abstract

In this Final Degree Project, the design and implementation of an alert system that warns of tweets containing hate speech in their publication is presented. For this purpose, different types of models will be used, using techniques related to Automatic Learning, to make an automatic classification of these publications. These models will be explained and compared to choose the one that obtains better results on real and tagged tweets.

Key words: Detection, Hate Speech, Machine Learning, Twitter, Bot, Alert System, Python

Índice general

Índice general	VII
Índice de figuras	IX
Índice de tablas	XI

1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Estructura de la memoria	3
2 Marco teórico	5
2.1 Introducción	5
2.2 Técnicas	5
2.2.1 Pesado Tf-idf	6
2.2.2 Análisis de Componentes Principales	7
2.3 Modelos	8
2.3.1 K-Vecinos Más Cercanos	8
2.3.2 Máquinas de Soporte Vectorial	9
2.3.3 Redes Neuronales	10
3 Análisis del problema	15
3.1 Introducción	15
3.2 Clasificación de tweets	15
3.3 Datos	16
3.4 Tecnologías utilizadas	17
3.4.1 Python	17
3.4.2 Sklearn	18
3.4.3 Keras	18
3.4.4 Tweepy	19
3.5 Plan de trabajo	19
4 Diseño de la solución	23
4.1 Introducción	23
4.2 Sistema de alerta	23
4.2.1 Bot de Twitter	23

4.2.2 Modelo	24
5 Resultados	29
6 Relación con los estudios cursados	33
7 Conclusiones	35
7.1 Consecución de objetivos	35
7.2 Limitaciones	36
7.3 Trabajo futuro	37
Bibliografía	39

Apéndice	
A Matrices de confusión	41

Índice de figuras

2.1	Técnicas utilizadas en la extracción de características.	6
2.2	Clasificador KNN con $k = 5$	8
2.3	Espacio con muestras linealmente separables.	9
2.4	Transformación mediante Kernel	10
2.5	Desglose de la red LSTM	12
2.6	Funciones de activación más empleadas	12
3.1	Tweet tomado del dataset clasificado como <i>Non-Hate</i>	16
3.2	Tweet tomado del dataset clasificado como <i>Hate</i>	16
3.3	Topología de la red neuronal LSTM utilizada	21
4.1	Tweets predichos por el sistema de alerta como <i>Non-Hate</i>	27
4.2	Tweets predichos por el sistema de alerta como posibles <i>Hate</i>	27
4.3	Tweets predichos por el sistema de alerta como <i>Hate</i>	27
A.1	Matrices de confusión para el modelo KNN	41
A.2	Matrices de confusión para el modelo SVM	41

Índice de tablas

3.1	División del dataset	17
5.1	Resultados	30

CAPÍTULO 1

Introducción

Uno de los desafíos más importantes en el área del Procesamiento del Lenguaje Natural (*NLP* por sus siglas en inglés, *Natural Language Processing*) es la detección del análisis semántico en textos, es decir, el significado del texto. Dicho análisis es mucho más complejo de ejecutar que el análisis sintáctico, ya que éste último sigue unas reglas lógicas mientras que el primero carece de ellas. Debido a esta dificultad subyacente, el análisis del discurso de odio es un problema actual y relevante en este campo. Por consiguiente, mediante este trabajo se presenta una aproximación para abordar el problema.

En este capítulo de introducción, se enuncian los hechos que motivan la realización del presente trabajo, así como los objetivos del mismo. Por último, se describen, de manera más detallada, las secciones del trabajo que aparecen en el índice general.

1.1 Motivación

En primer lugar, cabe señalar que la elección del tema descrito en este Trabajo de Fin de Grado surge de la motivación personal e intrínseca hacia el campo de *Machine Learning* y la detección automática de patrones. Éstos son campos que han estado en auge en los últimos años. Asimismo, cabe añadir que otra de las razones que motivan este trabajo es el interés por aprender las diferentes técnicas que envuelven al Aprendizaje Automático, más en profundidad y enfocadas a un problema más real y significativo.

La dificultad del problema es otra de las motivaciones que impulsan la realización de este trabajo, ya que el problema que se presenta es mucho más real y complejo que las actividades que se puedan abordar en una sesión de prácticas

de una asignatura. Por ello, se han necesitado muchas horas de dedicación a la investigación para poder abordarlo, lo que ha permitido también un crecimiento académico y profesional en relación a este tema.

Además, Twitter es una red social muy utilizada en la actualidad. Ésta se caracteriza por su sencillez en las publicaciones, ya que son textos cortos de no más de 280 caracteres, pero con una gran y acentuada carga semántica. Gracias a ésta, se facilita en gran medida la clasificación de las publicaciones en Twitter según si contienen discurso de odio o no. Asimismo, debido a su popularidad en la sociedad, se pueden obtener una gran cantidad de muestras de ella, lo que puede suponer una ventaja en el momento de la realización del trabajo.

1.2 Objetivos

El objetivo general del presente trabajo es el diseño e implementación de un sistema de alerta frente a posibles publicaciones que contengan discurso de odio (en inglés, *Hate Speech*) en la red social *Twitter*.

Para la consecución del objetivo general se han diseñado una serie de objetivos específicos que se enumeran a continuación:

- Enunciar y comparar, según su eficiencia, una serie de modelos de clasificación, que permitan la separación de un conjunto de publicaciones o *tweets* en dos conjuntos, uno formado por tweets que fomenten el discurso de odio y otro por tweets que no lo hagan.
- Seleccionar, de entre los modelos comparados, el que haga la clasificación de una manera más óptima.
- Implementar un bot que lea en tiempo real, mediante un streamer, los tweets que pasen una serie de filtros como puede ser el idioma o que la publicación contenga un conjunto de palabras.
- Emplear diversas librerías externas para llevar a cabo el desarrollo del sistema de alerta.

1.3 Estructura de la memoria

En este apartado de la memoria se exponen, de un manera más extensa, los capítulos de los que consta la misma. Existen 6 capítulos que son explicados a continuación, excluyendo el actual que introduce el presente trabajo.

En el capítulo 2, Marco Teórico, se presentan las técnicas y modelos que son empleadas en el trabajo.

En cuanto al problema a tratar, en el capítulo 3, se realiza un análisis del mismo. También, se explican tanto los datos utilizados como las librerías y el lenguaje de programación utilizado para resolverlo.

En el capítulo 4, se muestra el diseño final que se ha realizado para conseguir una solución al problema. En este capítulo se expone todo lo implementado en relación a las dos partes del problema.

En el capítulo 5, Resultados, se interpretan, para cada modelo explicado, los que se han obtenido en las pruebas. Observando los resultados expuestos, se puede comprobar qué modelo es el que mejor funciona para así implementarlo como parte de la solución.

Las relaciones entre los conceptos aplicados en este trabajo y las asignaturas que han sido cursadas se exponen en el capítulo 6, siendo explicadas las competencias de cada asignatura.

Para finalizar el trabajo, en el capítulo 7, se analiza el cumplimiento de los objetivos tanto principales como específicos enumerados en la sección 1.2, así como también se enumeran las limitaciones que han surgido durante la elaboración del trabajo y futuras propuestas que podrán ser implementadas en caso de continuar con el trabajo.

CAPÍTULO 2

Marco teórico

2.1 Introducción

Como ya se ha indicado en la introducción del presente trabajo, el esfuerzo de investigación ha sido muy importante, por lo que se han utilizado diversos tipos de técnicas (en la extracción de características) y de modelos (para la clasificación de las muestras) para obtener una solución al problema, la detección del discurso de odio en tweets recuperados de la red social Twitter.

Por ello, en este capítulo se expone el Marco Teórico del trabajo. En él se presentan todos los tipos de técnicas empleadas en el trabajo para encontrar una solución al problema propuesto.

El capítulo actual se divide en dos grandes secciones: **Técnicas** y **Modelos**. La primera contiene todos los tipos de procedimientos utilizados para obtener una muestra cualitativa en una muestra cuantitativa, mientras que la segunda abarca todos los modelos sobre los que se han hecho pruebas en este trabajo a la hora de la clasificación de dichas muestras.

2.2 Técnicas

A continuación, se detallan las técnicas que han sido utilizadas para llevar a cabo la realización del trabajo.

En primer lugar, cabe mencionar que estas técnicas están principalmente enfocadas a la parte de extracción de características cuyo objetivo es convertir una muestra con datos cualitativos (como puede ser el texto de un tweet) en datos cuantitativos, es decir, un vector numérico que represente a la muestra y que

pueda ser utilizado en la clasificación. Para llevarla a cabo, se emplea la técnica llamada pesado Tf-idf (Figura 2.1) que es explicada más adelante, en la sección 2.2.1.

Otro procedimiento que también es empleado durante la extracción de características es el llamado Análisis de Componentes Principales o *PCA* (Figura 2.1). Su objetivo es el de reducir la dimensionalidad de las características de las muestras. Este método se explica en la sección 2.2.2.

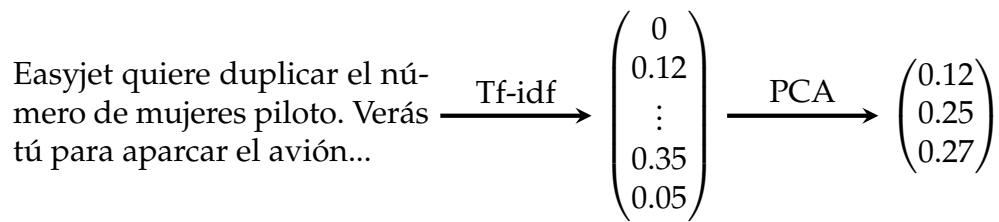


Figura 2.1: Técnicas utilizadas en la extracción de características. Tf-idf convierte el texto del tweet a un vector de números con D_1 dimensiones, y PCA convertirá ese vector en otro con $D_2 \leq D_1$ dimensiones.

2.2.1. Pesado Tf-idf

Como ya se ha indicado anteriormente en cuanto a la extracción de características, la técnica utilizada es el pesado Tf-idf (por sus siglas en inglés, *Term frequency - Inverse Document Frequency*). Esto se debe a que dicho método es muy eficaz, y por tanto muy usado, en el campo de la clasificación de texto con *Machine Learning*, como se puede comprobar en varias de las implementaciones del paper del concurso SemEval-2019 [3].

Este procedimiento consiste en la transformación de un fragmento de texto en un vector que contiene pesos. Cada elemento del vector puede verse como el peso o relevancia que tiene una palabra concreta en el texto. Este peso se calcula a partir de dos conceptos, que le dan el nombre al procedimiento: la **frecuencia de término** y la **frecuencia inversa de documento**.

La **frecuencia de término** o *term frequency* (cuyas siglas son tf) se suele calcular mediante la frecuencia de aparición de un término normalizada, es decir, como la cantidad de veces que aparece dicha palabra en el documento dividida por el número de apariciones de la palabra que más aparece en dicho documento

(Ecuación 2.1). Este cálculo indica qué palabra aparecerá con más frecuencia en el documento.

$$\text{tf}(t, d) = \frac{f(t, d)}{\max_{t \in d}(f(t, d))} \quad (2.1)$$

La otra parte de la técnica del pesado Tf-idf, la **frecuencia inversa de documento**, es utilizada para reducir el peso que tendrá una palabra si ésta aparece en muchos documentos, ya que se convertirá en una palabra menos relevante. Esta parte se calcula como el logaritmo de la división del número de documentos totales entre el número de documentos en los que aparece el término (Ecuación 2.2). Según este cálculo, en cuantos más documentos aparezca el término t , menor será $\text{idf}(t, D)$.

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (2.2)$$

Por último, el peso tf-idf final de un término sobre un documento se calcula según la ecuación 2.3, multiplicando los dos términos anteriores, que indica como de relevante es dicha palabra en ese documento en concreto.

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) \quad (2.3)$$

2.2.2. Análisis de Componentes Principales

Como se ha explicado en los párrafos introductorios de la sección 2.2, otra de las técnicas utilizadas en la parte de extracción de características es la llamada Análisis de Componentes Principales (PCA por sus siglas en inglés, *Principal Component Analysis*). Este procedimiento se basa principalmente en la reducción de dimensionalidad de las muestras.

La característica más importante de esta técnica es la elección de un nuevo sistema de coordenadas para las muestras, de tal manera que dicho nuevo sistema conlleve a la menor pérdida de información pero a su vez permita la reducción de dimensionalidad que se pretende obtener.

Este sistema se construye según las varianzas de las dimensiones del sistema actual, de tal manera que la primera dimensión del nuevo sistema será la dimensión con más varianza, la segunda será la segunda dimensión con más varianza...

y así sucesivamente. De esta manera, se obtiene un nuevo sistema con las dimensiones que contengan la mayor varianza de los datos, reduciendo así la pérdida de información al mínimo.

2.3 Modelos

En esta sección se presentan los diferentes tipos de modelos de clasificación del campo de *Machine Learning* que han sido empleados para la realización de este trabajo.

Esta sección se encuentra dividida en tres subsecciones, cada una dedicada a uno de los diferentes modelos utilizados en el presente trabajo. Estas subsecciones son: **K-Vecinos Más Cercanos**, **Máquinas de Soporte Vectorial** y **Redes Neuronales**.

2.3.1. K-Vecinos Más Cercanos

El modelo K-Vecinos Más Cercanos (*KNN* por sus siglas en inglés, *K-Nearest Neighbours*) es uno de los modelos más sencillos y, a la vez, potentes en el ámbito del *Machine Learning*. Este modelo pertenece a la familia de los Modelos basados en memoria, es decir, sin parámetros que aprender, que clasifican las nuevas muestras comparándolas con las anteriores.

Este modelo se basa en un principio muy básico: Una muestra será clasificada en la clase c si, entre las k muestras más cercanas a ésta en un espacio D -dimensional, hay mayoría de muestras de la clase c frente a cualquier otra clase c' . Como ejemplo, observando la Figura 2.2, se puede comprobar que para la muestra m_1 y para $k = 5$, se obtienen 3 muestras de la clase roja y 2 muestras de la clase azul, por lo que la muestra en cuestión sería clasificada como roja.

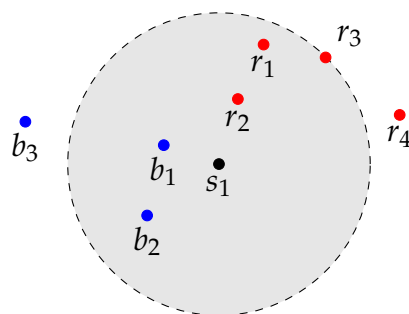


Figura 2.2: Clasificador KNN con $k = 5$.

2.3.2. Máquinas de Soporte Vectorial

Las Máquinas de Soporte Vectorial (*SVM*, que proviene del inglés *Support Vector Machines*) son un tipo de modelo también muy potente en este campo.

Los *SVM* se basan en la búsqueda de un hiperplano con la capacidad de separar de manera óptima las muestras de una clase frente a todas las muestras las demás. Además, que separe los datos de manera óptima significa que dicho hiperplano maximizará su distancia con respecto a los puntos a separar.

Si se encuentra una situación en la que las muestras de las clases son linealmente separables (Figura 2.3), situar dicho hiperplano para dividir el espacio es relativamente sencillo. La complejidad radica en si el espacio de las muestras no puede dividirse mediante un hiperplano para separar dichas clases. Aquí es donde se utilizarán los llamados **Kernels**.

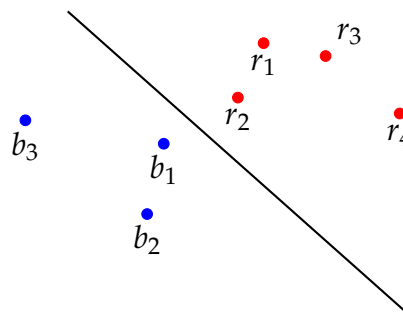


Figura 2.3: Espacio con muestras linealmente separables.

Los Kernels, referidos a ellos con ϕ , tienen como función principal trasladar el espacio D -dimensional en el que se encontraban las muestras a otro de mayor dimensión en el que los datos sean linealmente separables (Figura 2.4). De esta manera, se sitúa el hiperplano en un espacio de más dimensiones que el original, encontrando así una frontera de decisión no lineal en el espacio original para los datos.

Si aún así se obtuviese un espacio en el que las muestras no son linealmente separables, se utilizaría un parámetro del modelo llamado **slack**, que puede entenderse como la cantidad de muestras que pueden ser clasificadas incorrectamente o que se sitúan cerca de la frontera de decisión durante el entrenamiento del modelo, con tal de conseguir una clasificación más fiable y evitar que el modelo sobre-entrene. Este parámetro se puede usar, por ejemplo, cuando las muestras de las clases contienen ruido y se quiere obviar cierta cantidad de muestras que son anómalas.

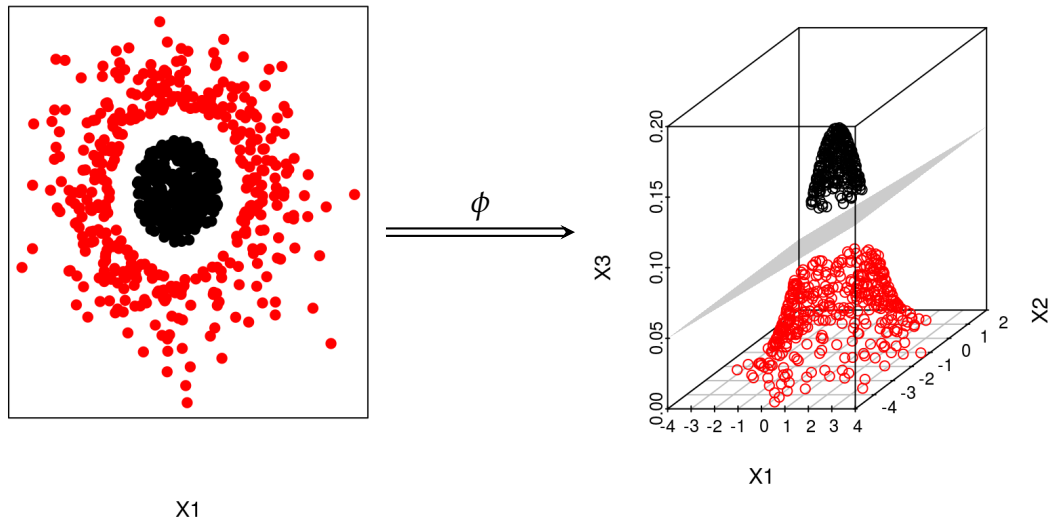


Figura 2.4: Transformación mediante Kernel de un espacio \mathbb{R}^2 no linealmente separable a un espacio \mathbb{R}^3 linealmente separable. **Fuente:** https://rpubs.com/Joaquin_AR/267926

2.3.3. Redes Neuronales

Durante estos últimos años, las redes neuronales artificiales han tomado una fuerza muy importante en el mundo del Aprendizaje Automático debido a la versatilidad que éstas ofrecen a la hora de implementar un modelo de clasificación.

Las redes neuronales tienen como parámetro una función llamada función de *loss*, que representa el error que este clasificador está cometiendo al hacer una predicción. Existen un gran número de funciones que se pueden utilizar como función de *loss*, la más sencilla y la que se utilizará en este trabajo es la de *Mean Squared Error*, que se calcula como la media de los errores entre las etiquetas (Ecuación 2.4).

$$l = MSE = \frac{1}{n} \sum (Y - y)^2 \quad (2.4)$$

Las redes neuronales basan el entrenamiento de sus parámetros en la modificación de éstos en base a unos gradientes. Estos gradientes pueden entenderse como la pendiente de la función de *loss* con respecto de los parámetros de la capa en cuestión. En otras palabras, si calculamos el gradiente g tal que sea la derivada de l con respecto de un parámetro W ($g = \frac{\partial l}{\partial W}$), entonces simplemente modifi-

cando el parámetro de la siguiente manera $W = W - \eta \cdot g$ se obtendrá un nuevo parámetro W con el que la función de *loss* se verá disminuida¹.

Conociendo el funcionamiento de las redes de una manera más general, a continuación, se profundiza un poco más en cada una de las capas.

Embeddings

Las capas de Embedding son unas de las más utilizadas en el campo del Procesamiento del Lenguaje Natural.

La función de una capa de Embedding no es más que la de convertir un dato cualitativo, como puede ser una palabra, en un vector de características con D dimensiones. La peculiaridad de este tipo de capas, como se explica en el paper *Why do we use word embeddings in NLP?* [6], es la de obtener unos vectores de características tal que dos palabras que tengan un significado semánticamente similar, obtengan vectores que se sitúen cerca en el espacio de dimensiones.

De esta forma, si se tuviese una red con una capa de Embedding y se intentase convertir las palabras "gato", "perro" y "negro" obtendríamos lo siguiente. "gato" y "perro" tendrían unos vectores similares, es decir, se situarían cerca en el espacio, mientras que el vector para "negro" con cualquiera de los otros dos, sería muy diferente.

LSTM

Los modelos LSTM, que provienen del inglés *Long short-term memory*, son un tipo de modelos que se denominan recurrentes, esto se debe a que se aplican una y otra vez las mismas operaciones sobre datos distintos, pero almacenando o teniendo en cuenta los datos anteriores. El interior, es decir, las operaciones de una de las celdas LSTM, se pueden comprobar en la figura 2.5b, así como observar como una misma celda se aplica repetidamente en la figura 2.5a.

Ésta es la característica principal de los modelos LSTM y la gran diferencia con los modelos recurrentes básicos. Estos modelos básicos solo son capaces de almacenar información a corto plazo, en otras palabras, solo tendrán en cuenta el elemento que han calculado en el momento inmediatamente anterior. Sin embargo, con los modelos LSTM se tiene en cuenta información que se ha calculado con

¹ η es el factor de aprendizaje, que se decide mediante pruebas sobre el modelo, escogiendo el que mejores resultados obtenga.

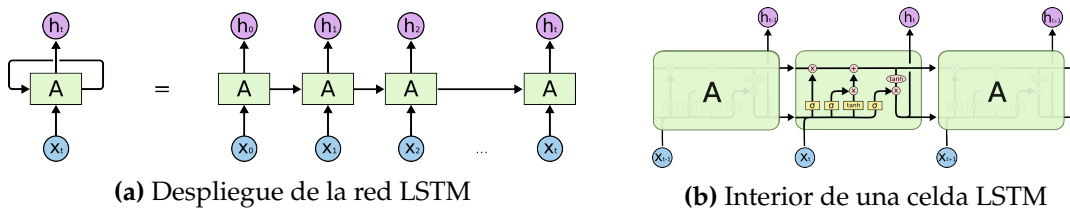


Figura 2.5: Desglose de la red LSTM.

Fuente: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

mucha más antelación, lo que permite a la red hacer una clasificación basándose en todos los elementos anteriores.

Densas

Por último, la última capa y la más sencilla que resta por explicar es la capa Densa o, también llamada, *Fully Connected*. El nombre se debe a que es una capa formada por un número N de neuronas y están todas conectadas tanto con la capa anterior como con la siguiente.

Este tipo de capas son las más básicas dado que simplemente aplican una transformación lineal con la forma $y = Wx + b$; donde x es el vector que proviene de la capa anterior, W es la matriz que contiene los parámetros de dicha capa y b es el vector desplazamiento que también será aprendido durante el entrenamiento.

Finalmente, se aplica una función, llamada función de activación, a dicha transformación lineal. Las más usuales son ReLU, Sigmoide o TanH, que pueden ser comprobadas en la figura 2.6.

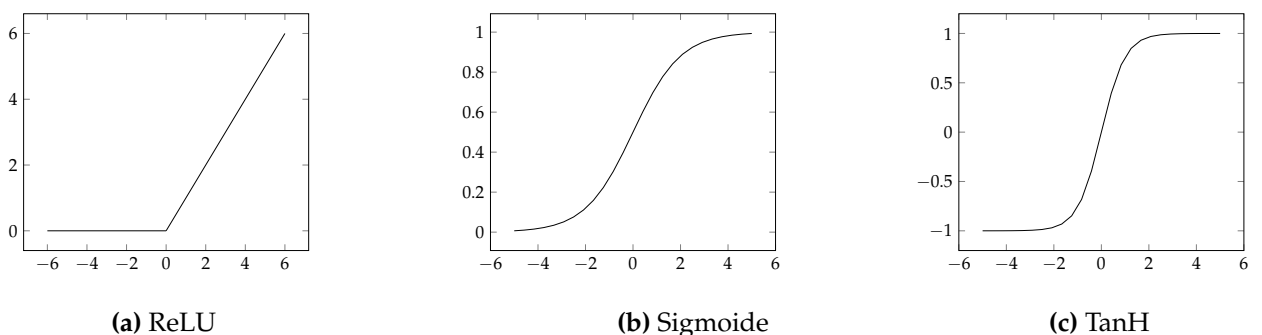


Figura 2.6: Funciones de activación más empleadas

Una funcionalidad añadida que tienen este tipo de capas es el llamado *Dropout*. Este concepto tiene como finalidad reducir el sobre-aprendizaje o *overfitting*, ya que durante la sesión de entrenamiento de la red, algunas neuronas devolverán 0, con una probabilidad p , en vez de su valor calculado, de tal forma que la capa

siguiente deberá seguir aprendiendo aún con datos faltantes. Las probabilidades p que se asignan para que ocurra el *dropout* en una neurona son bajas, se suelen utilizar valores que rondan el ~30 %.

CAPÍTULO 3

Análisis del problema

3.1 Introducción

Este nuevo capítulo, el cual se encuentra dividido en cuatro partes, muestra un análisis más exhaustivo y profundo del problema a tratar.

En primer lugar, se explica en qué consiste el problema en su totalidad al cual se ha querido dar una solución en este trabajo, así como de donde procede la idea. Seguidamente, se explican los datos utilizados tanto en el entrenamiento de los modelos como en la parte de obtención de resultados. Además, se exponen las tecnologías como lenguajes de programación, librerías... que han sido utilizadas en la implementación de todas las partes del trabajo. Por último, se expone el plan de trabajo que se ha seguido durante el tiempo de realización del mismo.

3.2 Clasificación de tweets

La clasificación de textos es un problema muy presente en el campo del *Machine Learning*. En el presente trabajo se pretende dar una solución a un problema de este campo, más concretamente, la clasificación del discurso de odio y agresividad en publicaciones de la red social Twitter.

Asimismo, a parte del modelo que se encargará de la clasificación de dichos tweets, se pretende implementar un sistema de alarma mediante un bot de Twitter que lea los tweets que se publiquen en tiempo real y los clasifique utilizando el modelo, lanzando una alarma en caso de que el tweet contenga discurso de odio.

3.3 Datos

En cuanto a los datos utilizados en el presente trabajo, tanto a la hora de entrenar los clasificadores como en la parte de la obtención de los resultados, se ha utilizado el dataset empleado en la competición SemEval del año 2019, más concretamente en la tarea 5 [3] referida a la detección del discurso de odio dirigida hacia inmigrantes y mujeres en Twitter.

Dicho dataset tiene por nombre *HateEval* y consiste tanto en tweets en inglés (unos 13.000 tweets) como en español (unos 6.600 tweets), etiquetados con un 1 (*Hate*) si el tweet en cuestión promueve un discurso de odio, o 0 (*Non-Hate*) en caso contrario. Además, contienen otras etiquetas (que solo existen si el tweet contiene discurso de odio), como *Aggressiveness* que indica si el tweet es agresivo o no, o *Targeted* que marca si el odio del tweet se encuentra focalizado en una persona en concreto o en un colectivo más genérico.

Algunos ejemplos de tweets, clasificados como *Non-Hate* o *Hate*, pueden verse en las figuras 3.1 y 3.2, respectivamente.


	Mira @██████████ en un futuro busca trabajos en los que te quieran PORQUE NO MERECEES ESTA PUTA MIERDA DE TRATO	<i>HS</i> 0	<i>TR</i> 0	<i>AG</i> 0
---	--	----------------	----------------	----------------

Figura 3.1: Tweet tomado del dataset clasificado como *Non-Hate*


	Buenos días, a trabajar en la fabrica para pagar a los refugiados y demás... gracias Sánchez por ser tan hijo de puta.	<i>HS</i> 1	<i>TR</i> 0	<i>AG</i> 1
---	--	----------------	----------------	----------------

Figura 3.2: Tweet tomado del dataset clasificado como *Hate*

El dataset se encuentra dividido en tres subconjuntos: *Training*, *Validation* y *Testing*. En la tabla 3.1 se puede observar la distribución existente de las muestras entre los distintos subconjuntos y las diferentes etiquetas de clase.

Como se puede observar gracias a la tabla 3.1, el dataset se encuentra suficientemente equilibrado en cuanto a las dos clases existentes. Se puede comprobar que la distribución entre las muestras que han sido clasificadas como *Hate* y las que han sido clasificadas como *Non-Hate* siguen unos valores 40 %-60 %.

Tabla 3.1: División del dataset

	Español			Inglés		
	Non-Hate	Hate	Total	Non-Hate	Hate	Total
Training	2631 (59 %)	1838 (41 %)	4469 (68 %)	5217 (58 %)	3783 (42 %)	9000 (69 %)
Validation	278 (56 %)	222 (44 %)	500 (8 %)	573 (57 %)	427 (43 %)	1000 (8 %)
Testing	940 (59 %)	660 (41 %)	1600 (24 %)	1719 (58 %)	1252 (42 %)	2971 (23 %)
Total	3849 (59 %)	2720 (41)	6569 (100 %)	7509 (58 %)	5462 (42 %)	12971 (100 %)

A pesar de esta división, que pueda parecer ligeramente desequilibrada, cabe destacar que en Twitter el número de tweets con discurso de odio será muy inferior a tweets sin él.

Podemos observar que el conjunto de *Training* tanto en español como en inglés se encuentra alrededor de un $\sim 70\%$, mientras que el de *Testing* es mucho más bajo, sobre un $\sim 25\%$, permitiendo así un entrenamiento mucho más fiable, pero a su vez dejando suficientes muestras libres para obtener resultados.

En resumen, se puede comprobar observando las tablas, que los datos utilizados siguen una distribución bastante equilibrada en cuanto a las muestras por clase tanto en el dataset en español como en inglés. Además, se distribuyen de una manera correcta en *Training*, *Validation* y *Testing*.

3.4 Tecnologías utilizadas

Para poder implementar tanto el modelo como el bot de Twitter que forman parte de este trabajo, se han utilizado una serie de tecnologías muy empleadas en el campo del Procesamiento del Lenguaje Natural, que se enumeran y explican brevemente en el siguiente apartado.

3.4.1. Python

Python [10] es uno de los lenguajes de programación más utilizados actualmente en todo el mundo. Destaca en la programación de scripts debido a su versatilidad y sencillez. Asimismo, es un lenguaje de programación interpretado y de alto nivel, lo que permite que escribir programas sea más sencillo y claro.

Además, Python dispone de un sistema muy robusto, pero a su vez sencillo, en cuanto a la gestión y manipulación de cadenas de caracteres, característica que ha facilitado en gran medida la posible implementación del vigente trabajo.

Otra gran ventaja de utilizar este lenguaje de programación es la cantidad de librerías externas que existen para la implementación de modelos de *Machine Learning* de una manera simple pero eficaz.

Debido a todo lo enumerado anteriormente, Python (en su versión 3.7) ha sido el lenguaje escogido a la hora de programar los scripts del presente Trabajo de Fin de Grado.

3.4.2. Sklearn

Sklearn [8] (o, por su nombre completo, Scikit-Learn) es una librería ampliamente usada en Python para la implementación de soluciones relacionadas con el *Machine Learning* como pueden ser modelos de regresión o de clasificación.

Algunos de los modelos explicados anteriormente como el K-vecinos Más Cercanos en la sección 2.3.1 o la Máquina de Soporte Vectorial en la sección 2.3.2 han sido implementados gracias a esta librería.

Sklearn tiene una parte, además de todas las funciones para implementar modelos de *Machine Learning*, que permite hacer una implementación sencilla tanto del pesado Tf-idf (explicado en la sección 2.2.1) como de PCA (explicado en la sección 2.2.2). Ambas funciones de la librería han sido utilizadas en el presente trabajo.

3.4.3. Keras

Otra librería externa de Python que ha sido empleada en la implementación del presente trabajo es Keras [5]. Se ha usado la citada librería en la escritura de los scripts que hacen uso de redes neuronales.

Keras es una librería muy utilizada en la implementación en Python de redes neuronales ya que contiene una gran cantidad de tipos de capas para incorporar en las redes, como son las LSTM, los Embeddings o las Convolucionales. Algunas de esas capas han sido ya explicadas en la sección 2.3.3.

Asimismo, la ya nombrada librería contiene, entre sus múltiples funcionalidades disponibles, unas métricas que han permitido medir, usando diferentes criterios, el grado de eficacia de cada uno de los modelos que se ha implementado y probado.

3.4.4. Tweepy

Tweepy [9] es otra de las librerías utilizadas en la elaboración de la parte de programación del presente trabajo. Esta biblioteca ha sido empleada en la parte de la escritura del bot de Twitter. Éste envía ordenes a la API oficial de Twitter utilizando Tweepy y facilitando así la elaboración del mismo.

Tweepy se conecta en primera instancia con la API de Twitter mediante unos tokens propios de la cuenta mediante la que se tiene que conectar, como ocurre en cualquier API online.

Esta librería tiene dos modos de funcionamiento:

- **Streaming:** Consiste en un modo en el que el bot recibe todos los tweets que se publiquen en tiempo real y que cumplan una serie de requisitos incluidos en los argumentos como puede ser el idioma del tweet o que éste contenga una serie de palabras.
- **Lookup:** Modo por el cual se obtiene una cantidad fija de tweets que siguen unos parámetros, que no tienen por que haber sido publicados en este preciso momento.

3.5 Plan de trabajo

Por último, en el apartado final de este capítulo se explica cual ha sido el plan de trabajo que se ha pretendido seguir en la implementación de todas las partes del mismo.

Extracción de características

Inicialmente, la primera tarea que se ha decidido abordar es la parte de **extracción de características**.

Primero, se ha utilizado una librería llamada Pandas [7] para leer los archivos en formato *tsv*, que contenían las muestras del dataset explicado en la sección 3.3.

Estos ficheros incluían tanto los textos de los tweets como la clasificación propia de cada uno, según si promueven o no el discurso de odio, separados por un espacio de tabulador.

Una vez obtenida una lista con los textos y otra con las etiquetas de las clasificaciones, se pasó a la implementación tanto del pesado Tf-idf como de PCA, ambas utilizando la librería Sklearn. Para utilizar estas técnicas primero se entrenan ambas con el conjunto de *Training* para, más tarde, convertir los demás conjuntos de tweets a vectores (con Tf-idf) y reducir sus dimensiones a una cantidad $D = 4965$ (con PCA), para el dataset en castellano, y $D = 10000$, para el dataset en inglés. Esta dimensionalidad ha sido escogida según la recomendada por la librería Sklearn y es igual al número de palabras en el vocabulario.

Modelos

Una vez obtenidos los vectores de características que representan los tweets del dataset, el siguiente paso es implementar y probar diversos tipos de modelos que serán enunciados a continuación.

Para comenzar, el primero de los modelos implementados es el K-Vecinos Más Cercanos, con un parámetro $K = 75$. Éste es un modelo sencillo, pero muy potente, por lo que se espera que obtenga unos resultados importantes.

Observando los resultados del MEX-A3T [1], un evento para mejorar la investigación del campo del NLP, se comprueba que existen diversas soluciones que utilizan Máquinas de Soporte Vectorial o SVM y obtienen unos resultados notables. Por ello, se ha decidido probar dicho modelo en la implementación del presente trabajo y comprobar si éste consigue resultados similares.

Una vez probados estos dos modelos básicos en el campo de *Machine Learning*, se intentará hacer una nueva aproximación al problema, esta vez con un clasificador un poco más complejo: una red neuronal. La red que se quiere emplear hace uso del modelo LSTM, basándose en diversas implementaciones de [2] y en el modelo que se utiliza en [11], para mantener la información y el contexto en mayor medida entre las palabras de los tweets. La red que se pretende utilizar tiene la topología que se ve representada en la figura 3.3.

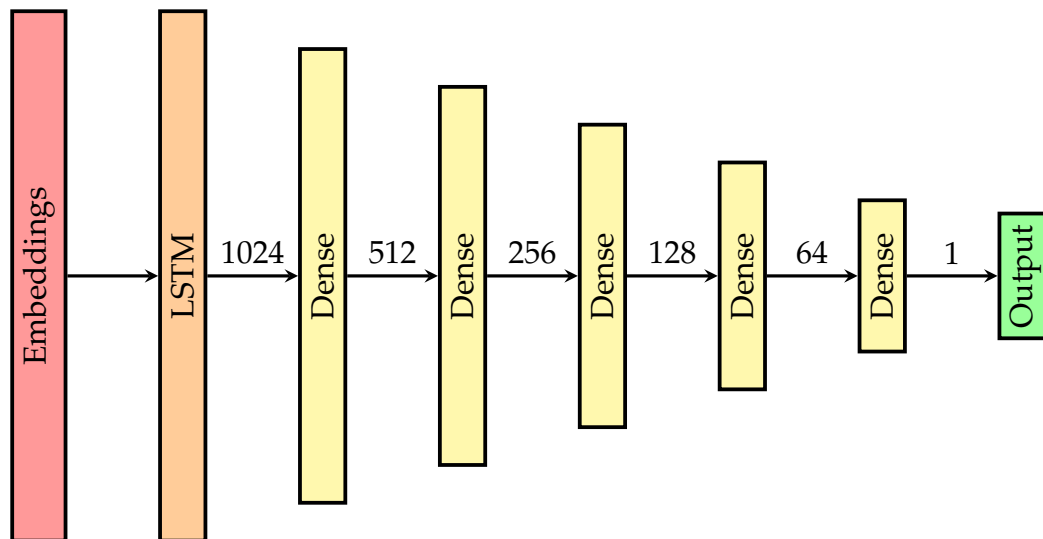


Figura 3.3: Topología de la red neuronal LSTM utilizada

Otras implementaciones añadidas

Otras utilidades que se añadieron a la implementación del código de este trabajo fueron las siguientes.

La funcionalidad para realizar el pesado Tf-idf mediante la librería Sklearn contiene un tokenizador por defecto, que separa las palabras de una frase de una manera estándar. Sin embargo, existe una librería en Python que contiene un tokenizador más especializado en la separación de las palabras contenidas en un tweet. Dicha librería es conocida como NLTK [4] (por sus siglas en inglés, *Natural Language ToolKit*) y la funcionalidad citada se denomina TweetTokenizer.

Para suplir la ortografía de los tweets que pudiesen estar mal escritos, se buscó una librería que permitiese corregir las palabras que se encontrasen con faltas de ortografía y sustituirlas por una palabra existente en el diccionario. HunSpell¹ y PySpellChecker² son dos librerías que permitían la corrección de dichas palabras.

Bot de Twitter

Por último, una vez implementadas todas las partes de la detección del discurso de odio, se pasará a la creación del bot de Twitter. La implementación de esta parte será explicada con más profundidad en la sección 4.2.1 más adelante.

¹Web oficial de HunSpell: <http://hunspell.github.io>

²Web oficial de PySpellChecker: <https://pypi.org/project/pyspellchecker>

CAPÍTULO 4

Diseño de la solución

4.1 Introducción

Una vez enunciado el plan de trabajo del proyecto con las etapas por las que debe transcurrir, se dispone a introducir cuál ha sido la solución final de este último.

Esta solución está redactada de manera que se explica de forma genérica, es decir, que no se especifica, por ejemplo, el modelo en concreto que se usará, sino todo lo que envuelve a éste. Estos detalles se resolverán con más detalle en la parte de Resultados, en la sección 5.

4.2 Sistema de alerta

Como se ha indicado anteriormente, el sistema de alerta que se pretende implementar consta de dos partes que serán explicadas a continuación. Estas dos partes son un **bot** de Twitter para obtener los tweets publicados y un **modelo** de *Machine Learning* entrenado y que clasificará dichos tweets.

4.2.1. Bot de Twitter

La idea principal del bot de Twitter es que éste lea todos los tweets que se publiquen en tiempo real y que cumplan cierto criterio de búsqueda (como el idioma en el que están escrito o que contengan una serie de palabras) para almacenarlos para su posterior análisis mediante el modelo.

Primeramente, se ha abierto una cuenta nueva en Twitter, ya que se buscaba una sin actividad para la implantación del bot en ella. Ésta es una cuenta de desa-

rrollador, que contiene una variedad de herramientas para obtener estadísticas. Además, será la que use el bot para leer todos los tweets que se publiquen.

Para la parte referente a la implementación del bot que se encarga de leer Twitter ha sido usada, como se indicó en la sección 3.4.4, la librería Tweepy de Python.

Para utilizar esta librería, primeramente se debe conectar con la API de Twitter utilizando un perfil. La cuenta de Twitter tiene asociado unos tokens personales de acceso que se le pasan como parámetro a la librería para que ésta cree la conexión.

Una vez conectada a la API con el bot, se ha utilizado una funcionalidad de la librería, llamada *Streaming*, para leer los tweets que sean publicados en tiempo real. Éstos podrán ser filtrados, ya sea porque contengan ciertas palabras que se indica en el parámetro *track* o porque las publicaciones se encuentren escritas en un idioma en concreto.

4.2.2. Modelo

A continuación, se presentan los modelos de clasificación, sin entrar en detalle de cuál es el que mejores resultados proporciona.

Primeramente, se centraron los esfuerzos en utilizar la librería Pandas, como se ha indicado con anterioridad, para la extracción de los tweets del archivo en formato *tsv* que conforma el dataset. Al utilizar esta librería, se extrajo la información del archivo en un tipo de estructura de datos llamado *dataframe* que consiste, básicamente, en una tabla. Dicho dataframe consta de las 4 columnas explicadas a continuación:

- **text:** En esta columna se encuentran los textos de los tweets.
- **HS:** En la columna HS (*Hate Speech*) se puede encontrar la etiqueta que indica si el tweet contiene discurso de odio (1) o no lo hace (0).
- **TR:** Esta etiqueta (*Targeted*) muestra si el tweet en cuestión está focalizado en una persona (1) o si se refiere a un colectivo (0).
- **AG:** La etiqueta AG (*Aggressiveness*) marca si el tweet emplea lenguaje agresivo (1) o no (0).

Una vez obtenido el citado *dataframe*, se ha abordado la implementación de toda la sección de la extracción de características de la siguiente manera. Creando el vectorizador Tf-idf que forma parte de la librería Sk-learn para transformar texto cualitativo en vectores numéricos, es decir, datos cuantitativos. El vectorizador Tf-idf tiene, por defecto, un tokenizador¹ básico, sin embargo, se ha decidido utilizar otro llamado TweetTokenizer debido a que hacía una mejor división de las palabras al hacerse un amplio uso de caracteres inusuales como '@' o '#'. Dicho vectorizador fue entrenado utilizando la función *fit*, de tal manera que las palabras más utilizadas en el dataset de *training*, tengan un peso mayor si aparecen en un tweet del dataset de *testing* o el de *validation*. Después, para conseguir el vector de características asociado al texto de un tweet, simplemente se debe ejecutar la función *transform*, que devolverá dicho vector.

Debido a la gran cantidad de palabras en el vocabulario, los vectores de características de los tweets tienen una dimensionalidad muy elevada como se indicó con anterioridad en la sección 3.5. Por ello, se debe implementar PCA, utilizando otra de las funcionalidades de Sk-learn, para reducir la dimensionalidad de las muestras a algo mucho más manejable. La herramienta de PCA funciona de una manera similar a lo descrito con anterioridad del vectorizador. Primero, se debe entrenar utilizando *fit* con la partición de *training* del dataset y utilizar el método *transform* para devolver el vector con la dimensionalidad disminuida según el número de dimensiones que haya aprendido de la partición de *training*.

Una vez se hayan obtenido los vectores de características con la dimensionalidad que se desee, el modelo deberá comenzar con la predicción. Para ello, primero se debe crear el modelo que se quiera emplear. En este caso, se utiliza el modelo que mejores resultados obtenga, que serán reportados en su correspondiente sección más adelante. En cualquier caso, después de haber creado el modelo con los parámetros que se requieran, se debe pasar al entrenamiento del mismo. Para ello, se utiliza de nuevo el método *fit* que forma parte del modelo para que éste aprenda a clasificar las muestras nuevas que le puedan llegar.

Una vez entrenado el modelo, éste debe comenzar a clasificar muestras que nunca haya visto el clasificador. Para ello, utilizamos una funcionalidad que está presente en los modelos creados con la librería Sk-learn que permite obtener la probabilidad de que una muestra en concreto sea clasificada con la etiqueta 1, es decir, que promueva el discurso de odio. Para darle más robustez al modelo, se clasificarán con un 1 todas las muestras que superen una probabilidad mayor a

¹Código que se encarga de dividir un cadena de caracteres compuesta por palabras en una lista con ellas.

un umbral θ , de esta manera todas las muestras para las que ocurra que $P(c = 0) \simeq P(c = 1)$, es decir, que son inciertas en su clasificación, serán clasificadas como si no promoviesen el discurso de odio.

Para resumir, matemáticamente, el modelo devuelve $P(c = 1)$ y la clasificación se lleva a cabo según la ecuación 4.1.

$$\begin{cases} 0, & \text{si } P(c = 1) \leq \theta \\ 1, & \text{si } P(c = 1) \geq \theta \end{cases} \quad (4.1)$$

También, se ha realizado una aproximación a una solución con la implementación de la funcionalidad para comprobar si una palabra no se encuentra en el diccionario o esté mal escrita. Se han programado dos pruebas de implementación con dos librerías diferentes, tanto con PySpellChecker como con HunSpell. El problema que se ha encontrado en estas implementaciones es que el tiempo de ejecución aumentaba notablemente y no se conseguían resultados muy concluyentes o, en algunos casos, incluso peores que con la implementación sin emplear este sistema, por lo que se descartó utilizar este tipo de correctores.

A continuación, se mostrarán algunas predicciones que ha realizado el sistema de alerta con tweets reales leídos mediante el stream del bot, utilizando una serie de palabras con connotaciones negativas para el filtrado, como son *asco*, *moro*, *inmigrante*...

Unas muestras de tweets que han sido clasificados como *Non-Hate* se encuentran en la figura 4.1; los tweets que han sido clasificados como *Hate* pueden comprobarse en la figura 4.3; y por último, algunas muestras han sido clasificadas como posibles *Hate* (es decir, que la probabilidad p de *Hate* es $0.5 \leq p \leq \theta$) y pueden verse dos ejemplos de este tipo de tweets en la figura 4.2.

Existe una pequeña variación que será explicada a continuación a la hora de entrenar la red neuronal enunciada en la sección 3.5.

La citada red comienza con una capa de Embedding y es por ello que no se necesita, cuando se utilice como modelo la red neuronal, toda la parte referente a la extracción de características, ya que esta primera capa es la encargada de hacer la extracción. Además, con esta capa, la red es capaz de aprender a extraer dichas características de mejor manera, debido a que la capa de Embedding sitúa palabras que tengan significados similares más cerca en el espacio vectorial, como se explica en la sección 2.3.3.



	Me he puesto súper triste de recordar ese momento y saber como son ahora todas las cosas. Es que vaya puta mierda que todo termine así...	<i>HS predicho</i> 0
	Este movimiento de @ [redacted] me parece una puta locura. Primer club de eSports que adquiere un club de fútbol profesional.	<i>HS predicho</i> 0

Figura 4.1: Tweets predichos por el sistema de alerta como *Non-Hate*



	@ [redacted] En qué mundo tu vives?! Tu no tienes vergüenza en esa cara!! No se les “ataca” por motivos políticos, ustedes atacaron al pueblo con la corrupción a costas del sufrimiento del pueblo! La única tragedia es que tú estes en ese puesto. Que pasó con tu transparencia? Das asco!!	<i>HS predicho</i> 1
	@ [redacted] @ [redacted] @ [redacted] Puta que verdad mas cierta, guzanos asquerosos, desaparecieron los wuevas a quienes estarán ayudando los comunachos cerdos??? Gracias mis carabineros!!!!	<i>HS predicho</i> 1

Figura 4.2: Tweets predichos por el sistema de alerta como posibles *Hate*



	Conchuda hija de puta	<i>HS predicho</i> 1
	@ [redacted] Bonito color el negro, en todo. Y muy negro en España, por esta gentuza	<i>HS predicho</i> 1

Figura 4.3: Tweets predichos por el sistema de alerta como *Hate*

CAPÍTULO 5

Resultados

En este capítulo se presentan los resultados obtenidos con los distintos modelos y cómo se han realizado las pruebas para obtenerlos.

Primeramente, se procede a aclarar como se han obtenido los resultados que se reportarán más adelante. Éstos han sido conseguidos con los tres modelos explicados en la sección 2.3, así como utilizando la vectorización comentada en el capítulo anterior y empleando PCA para reducir la dimensionalidad de las muestras. Los modelos son:

- Un modelo KNN con su parámetro $K = 75$
- Un modelo SVM con los parámetros por defecto que usa Sk-learn
- El modelo de red neuronal (NN, por *Neural Network*) explicado en la sección referenciada anteriormente, con la estructura de red que se puede observar en la figura 3.3.

Las muestras empleadas en la obtención de dichos resultados son las que se encuentran en la partición de *Testing*, tanto del dataset en español como el de inglés.

Los resultados obtenidos se miden utilizando 4 métricas muy empleadas en el campo de *Machine Learning* que son las siguientes. La primera utilizada es la *Accuracy*, en inglés, que se puede definir como el porcentaje de muestras clasificadas correctamente. La segunda se denomina *Precision* y se puede definir como, del total de predichos con etiqueta 1, cuántos tienen en realidad la etiqueta 1. La tercera métrica empleada se denomina *Recall* y se define como, de los que realmente están etiquetados como 1, los que el modelo predecirá con dicha etiqueta.

Por último, la cuarta métrica que se ha utilizado se llama *F1-score*, se calcula haciendo uso de la siguiente fórmula, $F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$, y se suele utilizar como una puntuación general del modelo. Estas tres últimas métricas, es decir, *Precision*, *Recall* y *F1-score*, están calculadas solamente para la clase 1.

Se puede comprobar los resultados desglosados en la tabla 5.1. Dicha tabla contiene datos para los tres modelos mencionados, con las cuatro métricas y para ambos datasets, tanto en inglés como en español.

Tabla 5.1: Resultados reportados según los diferentes modelos utilizados

	Español				Inglés			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
SVM	0.706	0.612	0.785	0.687	0.461	0.437	0.962	0.601
KNN	0.704	0.676	0.544	0.603	0.501	0.453	0.879	0.598
NN	0.413	0.413	1.000	0.584	0.421	0.421	1.000	0.593

Si se observan los resultados para la Red Neuronal, tanto en español como en inglés, se puede observar como este modelo no ha sido correctamente entrenado, ya que el valor del *Recall* a 1 indica que las predicciones de esta red para todas las muestras son *Hate* siempre. Por ello, se rechaza el modelo como uno de los posibles a utilizar en el sistema de alerta.

Observando la tabla, se puede comprobar como, para todos los modelos incluidos en ella, éstos funcionan peor para el dataset en inglés que para el de español.

Como también se puede observar en la tabla anterior, el mejor modelo que se ha conseguido es el SVM, dado que tiene los mejores resultados en la métrica de F1-score, que muestra de una manera más general el rendimiento del clasificador. Por ello, éste será el modelo que finalmente se use en el diseño final del sistema de alarma. Cabe resaltar que el *Recall* de este clasificador es más alto que el del modelo KNN, lo cual reduce la cantidad de Falsos negativos que se dan al hacer la clasificación, por lo que el modelo será más fiable.

Además, se han generado para los modelos SVM y KNN las matrices de confusión, a las que se les dedicará un capítulo en los apéndices, concretamente, en el A. Estas matrices están formadas por cuatro valores, que indican la cantidad de positivos verdaderos, falsos positivos, negativos verdaderos y falsos negativos.

A parte de comprobar cuál es el mejor de los modelos, también se puede observar claramente por las figuras A.1 y A.2, y reforzando los valores de la tabla 5.1, que el modelo funciona mejor en el dataset en español que en el inglés.

CAPÍTULO 6

Relación con los estudios cursados

En cuanto a los conocimientos técnicos que se muestran en el presente trabajo, gran parte de ellos han sido adquiridos gracias al grado en Ingeniería Informática y las diferentes asignaturas cursadas que forman parte del mismo.

En relación a los conocimientos generales de programación, cabe decir que las asignaturas de Introducción a la informática y a la programación (IIP) y Programación (PRG) son las que han aportado las bases para la obtención de los mismos. Asimismo, Sistemas de almacenamiento y recuperación de información (SAR) ha sido clave en la iniciación en el aprendizaje del lenguaje de programación llamado Python, el cual ha sido utilizado para llevar a cabo la implementación del sistema de alerta.

Por otra parte, los conocimientos matemáticos y todos los aspectos relacionados con el *Machine Learning* provienen de asignaturas como Percepción (PER), donde se situó el punto de partida de los futuros aprendizajes relacionados con la materia; Álgebra (ALG), mediante la que se adquirieron los fundamentos matemáticos que servirían para comprender cómo funcionan los modelos de *Machine Learning*; y, por último, Estadística (EST), a través de la que se profundizó en los conocimientos propios de esta materia, que mantienen una gran relación con el Aprendizaje Automático.

Gracias a todas estas asignaturas cursadas durante los cuatro años del grado, han podido adquirirse los aprendizajes necesarios para la realización del presente trabajo, y todos los apartados que en él se integran.

CAPÍTULO 7

Conclusiones

Finalmente, como ha podido comprobarse a lo largo de la realización del presente Trabajo de Fin de Grado existen múltiples herramientas que pueden ser utilizadas a la hora de implementar un sistema de alerta basado en modelos de Aprendizaje Automático.

A continuación, aparecen una serie de apartados a través de los cuales se expone si se han logrado los objetivos enunciados al inicio, se explicitan las limitaciones surgidas y cómo han podido superarse, y se enumera un conjunto de futuras propuestas que podrían llevarse a cabo para extender el contenido mostrado en el presente documento.

7.1 Consecución de objetivos

En relación al objetivo general que es el *diseño e implementación de un sistema de alerta frente a posibles publicaciones que contengan discurso de odio (en inglés, Hate Speech) en la red social Twitter* se ha conseguido a través de haber llevado a cabo los siguientes objetivos específicos.

Respecto al primer objetivo, el cual es *enunciar y comparar, según su eficiencia, una serie de modelos de clasificación, que permitan la separación de un conjunto de publicaciones o tweets en dos conjuntos, uno formado por tweets que fomenten el discurso de odio y otro por tweets que no lo hagan*, cabe mencionar que se ha logrado mediante la redacción del capítulo 5, Resultados.

En cuanto al objetivo específico siguiente, que dice *seleccionar, de entre los modelos comparados, el que haga la clasificación de una manera más óptima*, también se ha alcanzado a través del capítulo 5, Resultados, en el que se comparan diferentes

métricas para los modelos, eligiendo finalmente el modelo que ha obtenido los mejores resultados.

Cabe añadir, que la cumplimentación del objetivo específico que menciona el hecho de *implementar un bot que lea en tiempo real, mediante un streamer, los tweets que pasen una serie de filtros como puede ser el idioma o que la publicación contenga un conjunto de palabras*, se puede ver reflejado en la redacción de la subsección 4.2.1, donde se explica de una manera detallada cómo se ha ejecutado dicha implementación.

En cuanto al último objetivo específico que debía cumplirse, referido a *emplear diversas librerías externas para llevar a cabo el desarrollo del sistema de alerta*, puede verse logrado mediante la redacción de la sección 3.4, Tecnologías utilizadas, en la que se detallan las distintas características de las librerías que se han utilizado.

7.2 Limitaciones

Ante un trabajo de tal importancia y calibre es habitual que surjan limitaciones o dificultades que entorpezcan su realización. Por ello, cabe mencionar que las surgidas a la hora de elaborar los apartados del proyecto son diversas.

En primer lugar, cabe destacar el momento de la búsqueda de información. Al llevar a cabo esta acción nos hemos encontrado con dificultades propias de esta tarea, como son no obtener la información necesaria para cubrir los apartados estipulados en el índice o encontrar la información en fuentes que no eran fiables. Todo ello, han sido pequeños impedimentos que han podido ser solventados a medida que se avanzaba en el trabajo.

Otra de las limitaciones encontradas ha sido en cuanto a la capacidad de cómputo, dado que el ordenador sobre el que se estaba trabajando, en diversas situaciones, no contaba con la suficiente potencia o prolongaba demasiado el proceso de aprendizaje de ciertos modelos.

Cabe señalar, que una de las limitaciones presentes en el desarrollo del trabajo ha sido en cuanto a la presentación de una pequeña muestra de modelos, en comparación con la ingente cantidad existente. Pese a ello, se han seleccionado los que se consideraban más apropiados para el problema que se pretendía solventar.

7.3 Trabajo futuro

Finalmente, cabe subrayar que tras haber recabado información e implementado el sistema de alerta, son diversas las futuras propuestas de ampliación sugeridas, con el fin de mejorar el trabajo realizado.

De este modo, debido a la cantidad de datos que contiene el dataset y que no han sido empleados, como son las etiquetas de *Aggressiveness* o *Targeted*, se plantea la posibilidad de entrenar otros modelos para predecir dichas etiquetas y que el sistema de alerta sea más completo.

Otra futura propuesta podría ser, dado los modelos empleados en el desarrollo de este proyecto, ajustar los diferentes parámetros que sean propios de cada uno de tal forma que se consiga mejorar la eficacia del sistema de alerta, ya sea reduciendo el número de falsos positivos o disminuyendo el de falsos negativos para así poder obtener una serie de resultados más precisos y concluyentes.

Debido a la naturaleza del problema, otra propuesta consistiría en crear un portal en la web que mejorase la accesibilidad a los posibles tweets que el sistema de alerta marcarse como publicaciones que promueven un discurso de odio de cualquier tipo, de tal manera que se pudiesen revisar de una manera sencilla.

Bibliografía

- [1] ARAGÓN, M. E., ÁLVAREZ-CARMONA, M. Á., MONTES-Y GÓMEZ, M., ESCALANTE, H. J., VILLASENOR-PINEDA, L., AND MOCTEZUMA, D. Overview of mex-a3t at iberlef 2019: Authorship and aggressiveness analysis in mexican spanish tweets. In *Notebook Papers of 1st SEPLN Workshop on Iberian Languages Evaluation Forum (IberLEF), Bilbao, Spain* (2019).
- [2] BADJATIYA, P., GUPTA, S., GUPTA, M., AND VARMA, V. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion* (2017), pp. 759–760.
- [3] BASILE, V., BOSCO, C., FERSINI, E., NOZZA, D., PATTI, V., PARDO, F. M. R., ROSSO, P., AND SANGUINETTI, M. Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (2019).
- [4] BIRD, S., KLEIN, E., AND LOPER, E. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
- [5] CHOLLET, F., ET AL. Keras, 2015. <https://github.com/fchollet/keras>.
- [6] LATYSHEVA, N. Why do we use word embeddings in nlp? Towards Data Science. From <https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2>.
- [7] MCKINNEY, W., ET AL. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (2010), vol. 445, Austin, TX, pp. 51–56.
- [8] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

- [9] ROESSLEIN, J. Tweepy: Twitter for python! URL: <https://github.com/tweepy/tweepy> (2020).
- [10] VAN ROSSUM, G., AND DRAKE, F. L. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
- [11] ZHOU, C., SUN, C., LIU, Z., AND LAU, F. A c-lstm neural network for text classification, 2015.

APÉNDICE A

Matrices de confusión

En este apéndice se muestran las cuatro matrices de confusión de los experimentos realizados. Se puede observar las matrices del modelo KNN en la figura A.1 y las del modelo SVM en la figura A.2.

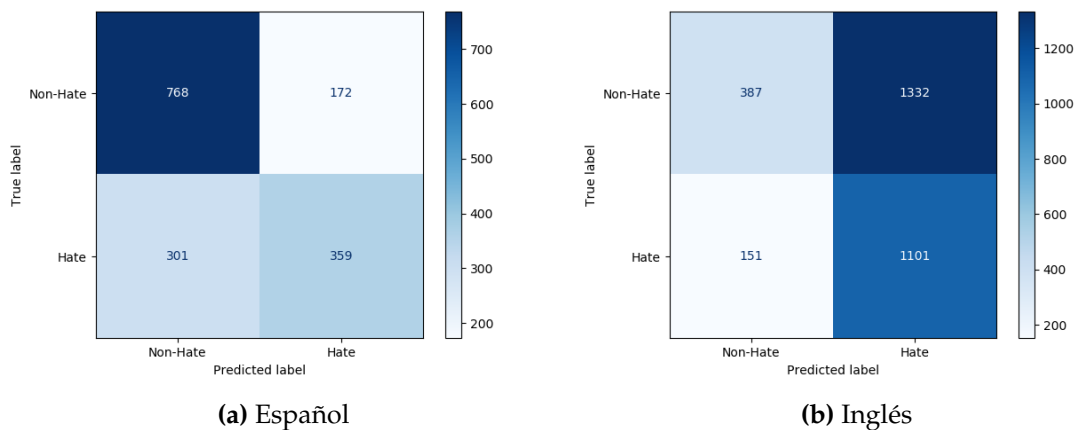


Figura A.1: Matrices de confusión para el modelo KNN

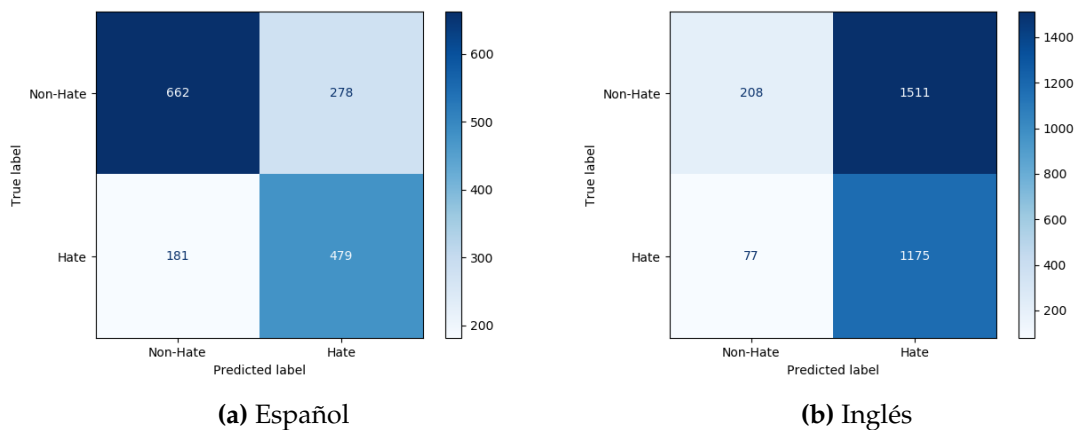


Figura A.2: Matrices de confusión para el modelo SVM