UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

CONSTRUCCIÓN DE MODELOS DE CLASIFICACIÓN AUTOMÁTICA PARA DISCURSOS DE ODIO

Juan Carlos Pereira Kohatsu
Tutor: Álvaro Ortigosa Juárez

JUNIO 2017

Construcción de Modelos de Clasificación Automática para Discursos de Odio

AUTOR: Juan Carlos Pereira Kohatsu
TUTOR: Álvaro Ortigosa Juárez

Grupo de Herramientas Interactivas Avanzadas (GHIA)
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen

Las llamadas *redes sociales* constituidas por plataformas tales como **Facebook™, Twitter™** constituyen el soporte de los *medios (de comunicación) sociales* que facilitan el intercambio y la discusión de información, experiencias y opiniones entre individuos de manera rápida y masiva.

La explosión de los medios sociales ha tenido consecuencias que han sido valoradas tanto positiva como negativamente para el conjunto de la sociedad.

Entre los efectos negativos, los medios sociales han hecho *'visibles'* algunas actitudes de ciertos grupos sociales que se traducen en ataques a personas o colectivos en razón de su pertenencia a unos determinados grupos definidos por características de nacionalidad, preferencias sexuales, raza, religión…que, en muchos países, han sido catalogados como delitos de odio (1)

Así pues, nace la necesidad de desarrollar un sistema que permita determinar si el autor de un mensaje es perpetrador de delitos de odio o no en una determinada red social, tarea nada sencilla de realizar puesto que la inmensa mayoría de los mensajes en las redes sociales no son de odio, lo que hace este problema muy similar al de buscar una *aguja en un pajar*.

Este proyecto toma como referencia la red social *Twitter* y los distintos tuits (*microblogs*) generados en la misma. Para crear un sistema de *detección de tuits de odio* en primer lugar será necesario desarrollar un modelo predictivo y posteriormente encapsularlo en un clasificador para que pueda utilizarse por el usuario final.

Durante el desarrollo del modelo ha sido necesaria la descarga de una gran cantidad de tuits haciendo uso de la API de Twitter y la posterior limpieza de estos, reduciendo la cantidad de ruido existente en los propios mensajes como repetición de caracteres y uso de símbolos extraños tales como los *emojis*. Además, se ha hecho uso de técnicas de procesamiento de lenguaje natural (NLP) permitiendo extraer información de los tuits previamente procesados, entre otras herramientas, ha sido necesario entrenar

Un analizador morfológico (*POS-Tagger*) para extraer clases de palabras que concentren la mayor parte de la semántica del mensaje (verbos, nombres y adjetivos).

Tras el tratamiento de los tuits, se ha desarrollado un filtro que permite equilibrar la cardinalidad de ambas clases de mensajes (odio y no odio) pasando de una proporción de 1:1000 a 270:1000. Una vez realizado el sobremuestreo de tuits con contenido de odio, se procede a la aplicación de técnicas de clasificación supervisada basadas en *aprendizaje máquina*, siendo las *redes neuronales profundas* el mejor clasificador para enfrentar este enmarañado problema. Finalmente, tras la validación del clasificador, se ha creado un conjunto de programas que permiten al usuario detectar *al vuelo* los tuits de odio.

Palabras clave

Aprendizaje máquina, colectivos, delitos de odio, discriminación, etnia, grupos sociales, medios sociales, nacionalidad, odio, orientación sexual, raza, red neuronal, religión, violencia, redes sociales, Twitter.

Abstract

The so-called “social networks” built-up by platforms such as **Facebook™, Twitter™** that operate on the Internet underpin the *social media* that facilitate quick and mass exchange and discussion of information, experiences and opinions between individuals. The explosion of social media has had consequences that have been valued both positively and negatively for society.

Among the effects considered as negative, social media have made 'visible' some attitudes of certain social groups that result in attacks on individuals or groups because of their affiliation to certain groups defined, by characteristics of nationality, sexual preferences, race, religion..., which in many countries have been classified as hate crimes

That is why it is necessary to develop a system to detect if a message in a social network contains hate speech. This is a very difficult task because most of the messages in social networks do not contain hate speech; this problem can be compare to find a *needle in a haystack*. This project will focus on *Twitter* and the small messages generated in this social network, also known as, tweets. In order, to develop this hate tweets detection system, first we build a predictive model that thereafter will be wrapped in a classifier easy to use by a final user. During this process, huge amounts of tweets were downloaded using Twitter’s API. After download, these tweets had to be cleaned to reduce the noise contained in them such as character repetitions and weird symbols like *emojis*. Furthermore, natural language processing (NLP) techniques have been used, specifically noteworthy, a morphological analyzer (POS-tagger) has been trained in Spanish to extract class of words, which contain most of the meaning a tweet (verbs, nouns and adjectives). After tweet cleaning, a filter has been applied in order to balance the number of tweets in both classes (hate and non-hate). As a result, the two class ratios changed from 1:1000 to 270:1000. On that oversampled tweets set, supervised classification techniques belonging to the machine learning have been used in the problem. A deep neural network has been selected as the most appropriate classifier to cope with this problem. Finally, a set of programs have been created that allow the user to detect on the fly the tweets that contain hate speech.

Keywords

Discrimination, ethnic group, hate crime, nationality, neural networks, race, religion, sexual orientation, social media, social networks, Twitter, violence

.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

Los *delitos de odio* son un tipo de infracción de la ley cuyo motivo principal es la existencia de *prejuicios* respecto a la víctima del mismo y tienen lugar cuando el perpetrador del delito elige a su víctima en base a su pertenencia a un cierto *grupo*.
Los atributos principales que definen el grupo de pertenencia de la víctima suelen ser el sexo, la etnicidad o raza, la nacionalidad, el idioma, la orientación sexual, la religión, la discapacidad, la apariencia física o la identidad de género, entre otros.
Existen evidencias de que tales delitos de odio están influidos por eventos *singulares* de *amplia difusión* (2) (atentados terroristas, migración incontrolada, manifestaciones, revueltas, etc.). Este tipo de sucesos suelen actuar como detonadores de manera que la frecuencia de este tipo de ilícitos aumenta espectacularmente tras ellos.
Los medios sociales de comunicación juegan un importante papel en la comisión de estos delitos en tanto y en cuanto las redes se llenan de mensajes de individuos afines a los perpetradores, que incitan a castigar al grupo elegido como diana. Estos mensajes, recogidos a lo largo de un periodo temporal posterior al incidente detonante, pueden servir para analizar la evolución de la amenaza: escalada, estabilización, duración y descenso.
Tal es la importancia de estos medios que, en muchos países, se han tipificado recientemente[1] también como pertenecientes a la categoría de delito de odio aquellas *manifestaciones públicas* que puedan considerarse una incitación al odio hacia ciertos colectivos. En España esto se conoce como **Discurso de Odio**.
Por estas razones, una herramienta capaz de detectar automáticamente discurso de odio sería de suma utilidad, y es el objetivo final de este trabajo.
Uno de los servicios más utilizados para realizar manifestaciones abiertas mediante la publicación de *microblogs* es Twitter™. Por este motivo este servicio se ha seleccionado como fuente básica de datos para el desarrollo de un sistema que, básicamente, **detecta** *lo que en este trabajo se denominan* ***tuits de odio***.
Como en todo proyecto relacionado con la *Ciencia de los Datos*, es evidente que, antes de trabajar con datos es preciso recogerlos, lo que se hará mediante la utilización de la API de Twitter.
A continuación, se realiza un análisis exploratorio de datos que servirá de base para la depuración de los mismos, su formateo y modelización.
A partir de los datos depurados, se procederá a su análisis mediante técnicas de *Procesado de Lenguaje Natural* (NLP) para extraer patrones y atributos de los textos para, finalmente, clasificar los mensajes mediante técnicas de *Inteligencia Artificial*(AI) como positivos (es decir, que representan un caso de discurso de odio) o negativos/neutros que no aportan pistas al respecto.
El elemento clave para tal clasificación es el **contenido del mensaje** en el que el redactor del mismo[2] –en este caso del *tuit*- manifiesta su sentimiento u opinión respecto a una *entidad* o aspecto de la misma.
El caso que nos ocupa, se enmarca dentro de un grupo de problemas de clasificación binaria - dos clases: contenido de odio o neutro - caracterizado por un ***desequilibrio*** muy

[1] En España en 2015 (1)

[2] O *fuente de opinión.*

pronunciado entre el número de instancias en cada clase[3], cuya proporción puede alcanzar valores próximos a 1:1000.
Este tipo de situaciones es de importancia en el mundo real en situaciones en que el coste de una clasificación errónea de las instancias de la clase *minoritaria* es muy elevado. Como ejemplos citaremos el diagnóstico de enfermedades o la detección de fraudes en tarjetas de crédito. En el primer caso, unos pocos píxeles del conjunto que constituye una imagen son la base del diagnóstico y en el segundo la proporción de fraudes sobre el total puede ser inferior al 1%. En el primer caso, un falso negativo puede incluso llevar a la muerte del paciente.
Esta situación de desequilibrio presenta tres problemas importantes para la clasificación de instancias:

1. Los mensajes de odio sobre un colectivo concreto *varían* a lo largo del tiempo *ligados a ciertos eventos* (atentado yihadista→mensajes antiislámicos, casos de corrupción→mensajes antipartidistas, premios Goya→cine español,…) lo que hace que los atributos relevantes para la clasificación sean variables y deban revisarse continuamente.
2. Se dificulta la aplicación de *métodos de clasificación supervisada* ya que para el etiquetado manual de unos cientos de casos, se requiere el examen de cientos de miles de tuits lo que alarga y encarece el etiquetado.
3. Por otro lado, el desequilibrio entre las clases provoca que el algoritmo que entrenamos sobre un conjunto con muy pocas instancias de la clase minoritaria sea, con frecuencia, incapaz de generalizar el comportamiento de esta clase y, por tanto, puede tener una escasa capacidad predictiva o una baja exhaustividad.
   Estos problemas, entre otros, se tratan en este trabajo.

1.1 Estructura

La memoria explicativa del proyecto se estructura de la siguiente manera:
Comenzamos revisando algunos *conceptos básicos* (§1.2) referidos a la clasificación en conjuntos con desequilibrio y revisaremos el *estado del arte* (§2) al respecto, así como las herramientas existentes para manejar el proyecto.
Seguidamente, en el apartado *diseño* (§3) se detallan aspectos fundamentales del mismo tales como

- Captura y limpieza de tuits,
- Representación de tuits,
- Selección para etiquetado y etiquetado manual
- Prueba y comparación de clasificadores,
- Validación y selección de clasificadores,
- Paquete de aplicación

A continuación, en desarrollo *del proyecto* (§4) se expone cómo se ha llevado a la práctica lo planeado en la fase de diseño, y tras comprobar y comparar los resultados obtenidos en *pruebas y resultados* (§5), se selecciona el clasificador con mejor desempeño que se encapsula para su utilización en el *sistema de explotación*. Por último en *conclusiones y trabajos futuros* (§6) se resaltan algunos puntos de importancia y se plantean posibles caminos de expansión y mejora del sistema.
Por último, se han incorporado Anexos que desarrollan algunos puntos de interés que, por limitaciones de espacio, en la memoria sólo son mencionados superficialmente.

[3] Es decir, diferencias significativas entre las probabilidades *a priori*.

1.2 Conceptos y problemas básicos

En este apartado expondremos algunos conceptos de particular interés para el desarrollo del TFG, algunos, más generales se encuentran en el (§Anexo A).

1.2.1 El ciclo de atención a un tema y el vocabulario

Como se ha mencionado anteriormente, los delitos de odio tienden a ser más frecuentes y a crecer en periodos de tiempo posteriores a un suceso antecedente ('detonador') (2) y el interés del público sobre un asunto determinado sigue el llamado *Ciclo de Atención a un Tema* que fue descrito inicialmente por Downs (4) y que se muestra en la Figura 0-1.

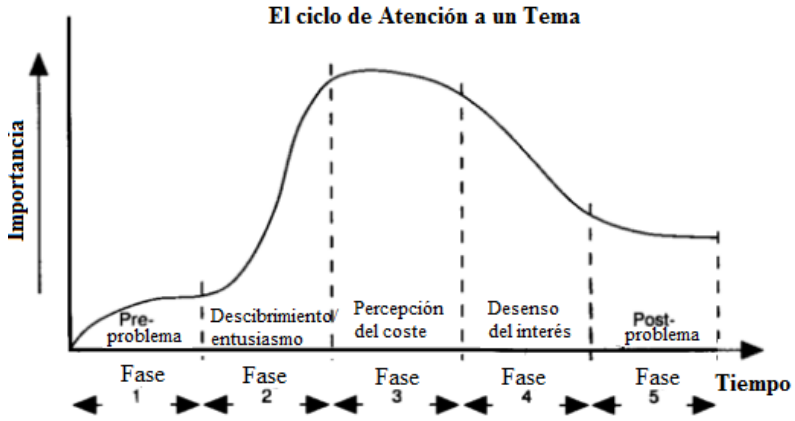


Figura 0-1: Ciclo de Atención a un Tema

Pueden verse las diferentes etapas por las que pasa la relevancia del tema para el público a lo largo de un periodo de tiempo que, además, no suele ser muy largo.
Este punto es importante por cuanto afecta tanto a la recolección de tuits como a los términos – atributos – a utilizar que se encuentran relacionados.
Sobre el primer punto, si se desea el seguimiento de un tema concreto una vez ocurrido un suceso (p.e. aparición de un nuevo caso de corrupción), hemos de recoger los tuits en la cresta de la ola. Además, hemos de tener en cuenta que, dependiendo de los asuntos que sean más *trendy*, el vocabulario que se usa para su comentario es diferente.

1.2.2 *Desequilibrio de clases, etiquetado y clasificación*

Nuestro problema de clasificación de tuits presenta un claro desequilibrio de clases, ya que la constituida por los tuits sospechosos de delito de odio, es mucho menos numerosa que la neutra (5). Este tipo de problemas se denominan '*la aguja en el pajar*'.
En aprendizaje supervisado, tenemos un conjunto – muestra – de datos $\mathcal{S}$ sobre el cual queremos construir un modelo de clasificación binario. La primera suposición que haremos es que las instancias del conjunto positivas y negativas observadas ($\mathcal{S}$ y $\mathcal{S}$) se extraen de dos distribuciones diferentes $\mathcal{P}$ y $\mathcal{G}$. Las instancias positivas son las minoritarias. En estas circunstancias es fácil ver que, por una parte, el etiquetado manual de las instancias resulta engorroso puesto que para encontrar un tuit – *la aguja*– de la clase minoritaria se precisa examinar una cantidad enorme de tuits de la otra clase – *la paja*; y, por otra, se explica el motivo por el cual un modelo

clasificador sobre el conjunto $S = C+ \cup C-$, siendo C+ la clase minoritaria y C- la mayoritaria, produce una baja exhaustividad: la distribución positiva está subrepresentada y los valores atípicos de $\mathcal{G}$– de mucha mayor cardinalidad – aunque sean una pequeña fracción, influirán en el clasificador ya que se considerarán por este como pertenecientes a C- y el clasificador inducido estará sesgado hacia la clase mayoritaria, es decir más cercano a los puntos de esta de lo que debiera, produciendo un rendimiento bajo del clasificador Figura 0-2: Errores de Clasificación en Muestras Desequilibradas (Figura 0-2(A)). La Figura 0-2 (B) muestra otro ejemplo cuando se usa 1-NN como clasificador en un conjunto desequilibrado. Lo mismo puede decirse de la clasificación bayesiana.

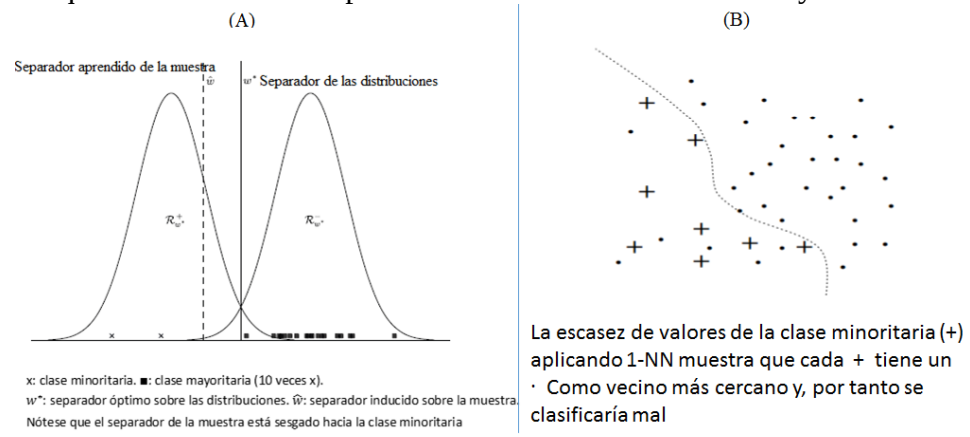


Figura 0-2: Errores de Clasificación en Muestras Desequilibradas

Debido a la dificultad de etiquetado manual, hemos utilizado un procedimiento que filtra el conjunto inicial extraído de Twitter mediante el uso de un vocabulario con términos de odio obtenidos de diferentes fuentes y el conjunto filtrado, de una cardinalidad mucho menor, se *etiqueta* y usa como conjunto de *entrenamiento* (Figura 0-3).

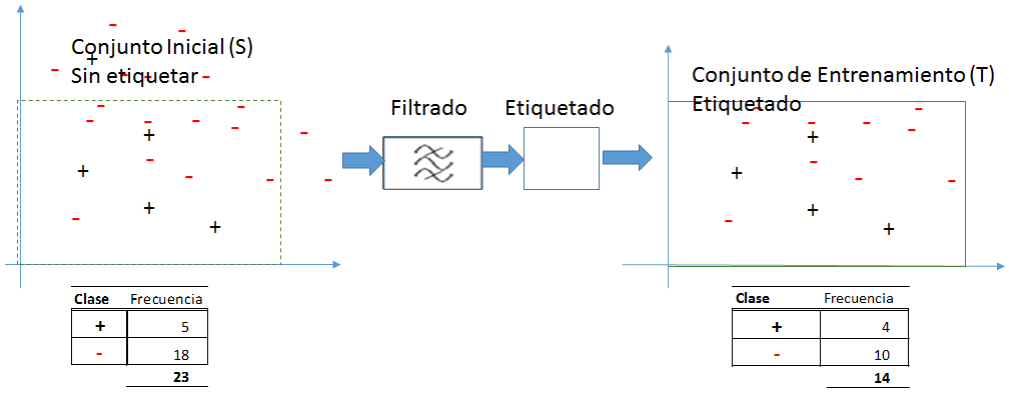


Figura 0-3: Filtrado del Conjunto Inicial

Seguidamente, el conjunto de entrenamiento se usa para estimar los parámetros del clasificador y, con este, se clasifican los tuits del conjunto de test.
Para la utilización del clasificador a otros conjuntos de tuits, caben dos opciones:

1. *Aplicarlo tras filtrar* el nuevo conjunto, asignando a todos los tuits que no pasan el filtro a *C-*.

2. *Aplicarlo al conjunto completo* con lo cual cabe la posibilidad de que algunos de los tuits que no pasan el filtro, se clasifiquen en *C+* (correcta o incorrectamente).

En el primer caso (Figura 0-4), la *matriz de confusión* muestra como FN a las instancias que no pasan el filtro que debieran estar etiquetadas con (+), una en nuestro ejemplo.

		Predicción		
		+	-	
Real	+	3	2	***5***
	-	1	17	***18***
		4	***19***	**23**

Precisión (p)	75,0%
Exhaustividad (r)	60,0%
F	66,7%

Tabla 0-1: Matriz de Confusión del Conjunto Inicial

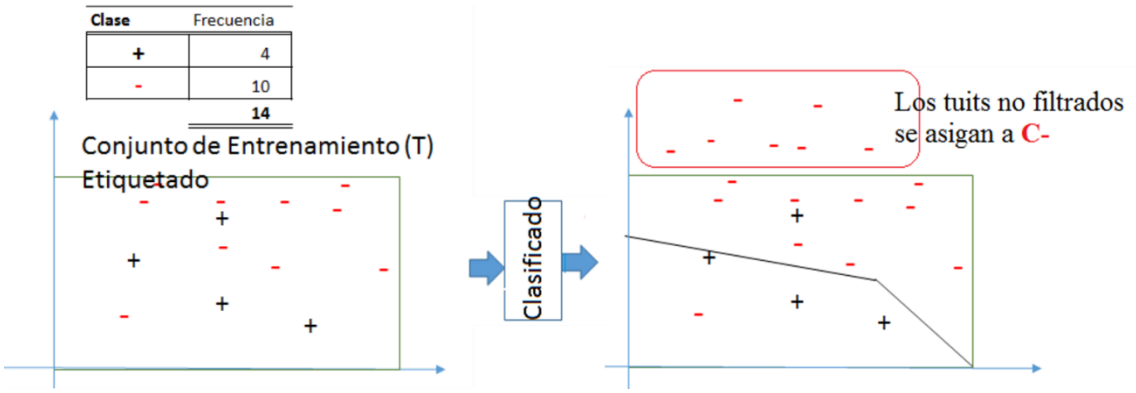


Figura 0-4: Clasificación tras Filtrado

Esta clasificación tras filtrado implica necesariamente una baja exhaustividad (§Anexo B).

El segundo caso es más complejo y, como veremos más adelante (§2.3), requiere un *recalibrado* de las probabilidades del modelo entrenado.

1.2.3 Clasificación supervisada o no supervisada

Hemos dado por sentado que utilizaremos métodos de clasificación supervisados sin haber justificado los motivos para el descarte de los no supervisados.

Podemos plantearnos una clasificación no supervisada, por ejemplo basada en análisis de conglomerados (*cluster).*

En este caso, nos enfrentamos a un problema de clasificación con clases *desequilibradas* y una de ellas debe ser la clase de *tuit de odio.*

Resulta imposible que, utilizando el procedimiento algorítmico k-medias (6) con *selección aleatoria de centroides iniciales* se clasifiquen las instancias en las clases de odio y neutras. Aumentaría esta posibilidad si asignásemos previamente uno de los centroides iniciales con unos valores aproximados al verdadero de la clase de odio, pero esto requeriría un *etiquetado previo* de las instancias que, en definitiva, es lo que hacemos directamente.

En cambio, cabría considerar la utilización del *clustering* tras el filtrado inicial para tratar de descubrir nuevas instancias que, de nuevo, serían etiquetadas a mano para aplicar finalmente un clasificador supervisado, proceso en cierto modo similar al de ampliar vocabulario a partir de los tuits de odio que usamos.

1.2.4 El Procesamiento del Lenguaje Natural en español

Un problema siempre presente en el PLN es que muchas de las herramientas existentes en los paquetes de uso libre están desarrollados inicialmente solo para el idioma inglés y, posteriormente, se han ido añadiendo versiones parciales para otros idiomas. Sin embargo, para el idioma español los recursos disponibles son escasos. Uno de estos recursos escasos lo constituyen los corpora ya que, mientras en inglés abundan corpora tanto escritos como hablados sobre diferentes asuntos (Project Gutenberg, Web chats, Corpus Brown Corpus Reuters, WordNet…) muchos de ellos anotados, es decir ya tratados por un *POS-Tagger*, lo que facilita en inglés el análisis semántico de los textos.

Es relativamente común en ciertos casos (sinónimos, análisis de sentimiento, etc.) proceder en tres etapas:

1. traducir de español a inglés,
2. utilizar bibliotecas para el inglés y
3. volver a traducir al español.

Como ejemplo de esto, en el Anexo E se muestra el proceso seguido para ampliar el vocabulario mediante sinónimos y algunos de los problemas que plantea.

Por esta razón, queremos resaltar que, tal como se expone en §4.1.3 ha sido necesario entrenar un *POS Tagger* en español utilizando para ello el Wikicorpus, v. 1.0 (7). Sin embargo, hemos de señalar que este *POS- Tagger* solo funciona en LINUX.

2 Estado del arte

Se trata de un trabajo que persigue la *clasificación* de una colección de tuits en las categorías:

- Odio
- No odio

Mediante procedimientos de clasificación *supervisada* de Aprendizaje Máquina, para lo cual debe procederse a un etiquetado previo de un conjunto de entrenamiento.

El proyecto se mueve, por una parte, en el terreno problemático delos conjuntos no equilibrados, tema sobre el que existe una abundante literatura que se refiere fundamentalmente a:

- cómo simplificar el etiquetado de las instancias y
- cómo paliar los efectos de la asimetría entre clases.

Por otro lado, el proyecto de clasificación de tuits utiliza herramientas del campo del *Procesamiento del Lenguaje Natural* (PLN),rama de la Inteligencia Artificial que tiene sus orígenes en los años 50 del pasado siglo cuando aparecieron las primeras computadoras. Su desarrollo ha venido acompasado a los avances en la capacidad de cómputo (*Ley de Moore*) y en los algoritmos de *aprendizaje estadístico.*

Por último, conviene pasar revista a las *herramientas informáticas* que facilitan el tratamiento de este tipo de problemas.

2.1 Detección de lenguaje de odio

Los intentos de aplicación de la inteligencia artificial a la detección de lenguaje de odio, son relativamente recientes y pueden seguirse en (8) y (9).

Dos interesantes situaciones que se repiten son:

1. Relacionar el mensaje con la fuente de manera que se detecten más mensajes de odio.
2. El etiquetado siempre se hace sobre un conjunto de mensajes bien
    a. de fuentes preseleccionadas (las de la etapa anterior), bien
    b. de mensajes prefiltrados.

Ya que, obteniendo una muestra aleatoria simple, resultaría prácticamente imposible el etiquetado.

2.2 Selección de instancias

Como hemos dicho, el *corpus* de tuits es un conjunto en el que la clase de tuits de odio es muy minoritaria respecto a la de tuits neutros. Este problema –junto con el de selección de instancias y desequilibrio de clases – ha sido estudiado desde muchos puntos de vista y T. Borovicka et al. (10) proporciona un resumen de los procedimientos desarrollados para seleccionar conjuntos de entrenamiento *equilibrado* en los casos en que existen grandes desequilibrios entre las clases de clasificación que básicamente son de tres tipos:

1. A nivel de datos
2. A nivel de algoritmo y
3. Conjuntos

Los primeros se basan en el sobremuestreo o submuestreo de la clase minoritaria o mayoritaria. Los segundos en ponderar de modo diferente la importancia de la instancia en función de la clase a que pertenezca y el tercero utiliza una combinación de métodos.

La fase inicial de *selección de instancias* es un proceso de reducir el conjunto de datos original.
La salida ideal de la selección de instancias es una muestra mínima independiente del modelo que pueda cumplir su objetivo con el menor deterioro posible, es decir, que el rendimiento $\mathcal{P}$ de un modelo $\mathcal{M}$ sea aproximadamente el mismo sobre la muestra S que sobre la población $\mathcal{W}$.

$$P(M_s) \approx P(M_w) \quad \text{Ec. (2-1)}$$

El método clásico de obtención de muestras se basa en las técnicas de muestreo (aleatorio simple, estratificado, adaptativo, …)
Con frecuencia, la muestra puede reducirse para generar un *conjunto de entrenamiento* más manejable (11).

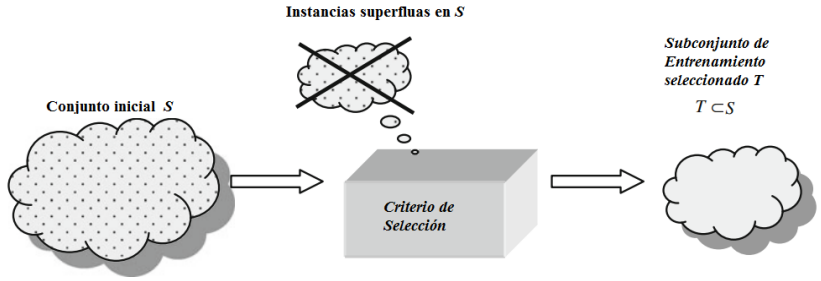


Figura2-1: Selección de Instancias

Esto puede hacerse utilizando algún tipo de algoritmo de *selección* de instancias bien relacionado con

1. el rendimiento de algún algoritmo de clasificación (*wrapper methods*) o con
2. el vector de atributos de la instanciación independencia del algoritmo utilizado (*filter methods*)

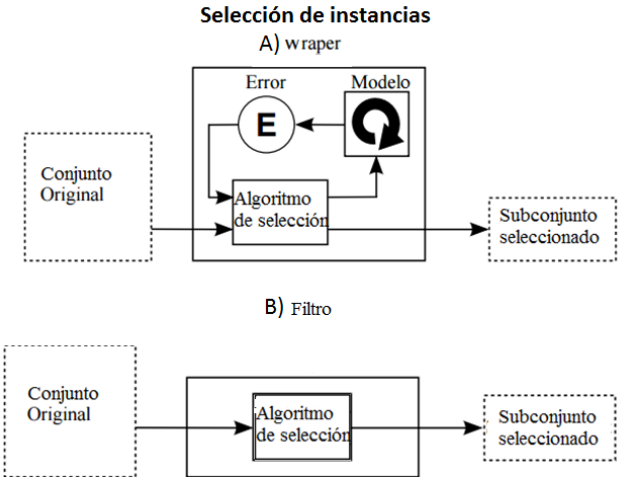


Figura2-2: Procedimientos de Selección de Instancias

Los enfoques del primer tipo (*wrapper*) subrayan el aspecto de minería de datos del problema ejecutando un algoritmo específico de tal campo para disparar la selección de instancias, p.e. seleccionando un subconjunto inicial, ejecutando un algoritmo sobre este subconjunto inicial, evaluando sus resultados y ampliando

incrementalmente el subconjunto inicial hasta que los resultados del algoritmo sean lo bastante buenos.
Los enfoques del tipo filtro, son más simples e independientes del algoritmo de clasificación.

2.3 Clases con probabilidades a priori no equilibradas

Hemos mencionado en §1.4 tanto la dificultad de etiquetar conjuntos con gran desequilibrio de clases como la de entrenar algoritmos en conjuntos de entrenamiento ya que ello conlleva un elevado *nivel de ruido* y un pobre rendimiento del clasificador.
P.e. si estamos realizando un experimento para el diagnóstico de alguna enfermedad cuya tasa de prevalencia es del 1‰ no elegiremos un conjunto de entrenamiento de 1 enfermo y 999 individuos sanos, sino un conjunto 50%-50%, donde los sanos constituyen el grupo de control y los enfermos el experimental, lo que modifica las probabilidades a priori de cada clase en el conjunto de entrenamiento e introduce sesgo en el clasificador.
En (10) se hace referencia a los siguientes grupos de métodos para vencer este problema:

1. Métodos de *nivel de datos*. Se utilizan en el preprocesado y se basan en varios tipos de *remuestreo*. Buscan aumentar el número de instancias de la clase minoritaria (sobremuestreo) y/o reducir los de la clase mayoritaria (submuestreo).
2. Métodos de *nivel de algoritmo*. Se basan sobre todo en dar una sobreponderación a la clase mayoritaria.
3. Métodos *conjuntos* que usan una combinación de métodos.

Todos ellos generan conjuntos de entrenamiento equilibrados sobre los que se entrenan algoritmos.
Resulta evidente que al reequilibrar artificialmente el conjunto de entrenamiento, las distribuciones en el conjunto de entrenamiento y en el de prueba son diferentes, de manera que se viola una de las hipótesis básicas de aprendizaje estadístico: que *tanto el conjunto de entrenamiento como el de prueba siguen la misma distribución*.
Esto motiva que la aplicación de un modelo entrenado sobre un conjunto de entrenamiento reequilibrado por el analista a un conjunto donde esto no se ha llevado a cabo (prueba) exige calibrar las probabilidades obtenidas del conjunto de entrenamiento retocado para poder aplicar el modelo recalibrado directamente al conjunto de test (no modificado).
En el artículo (12) se analiza este problema y se desarrolla un método para corregir el sesgo introducido.

2.4 Etiquetado de instancias

Un serio problema que afecta a los conjuntos desequilibrados, es el del *etiquetado del conjunto* de entrenamiento. Si bien resulta relativamente fácil recopilar cientos de miles de *tuits no etiquetados*, no es tan fácil proceder a su clasificación manual, especialmente cuando, como en este caso, la clase de interés es muy minoritaria.
Se han ensayado algunos métodos para vencer esta dificultad como el descrito en (13) que, a pesar de ser un método *no supervisado* está relacionado con la selección de instancias y ha servido de guía a nuestro proyecto y cuyo esquema es el siguiente (Figura2-3):

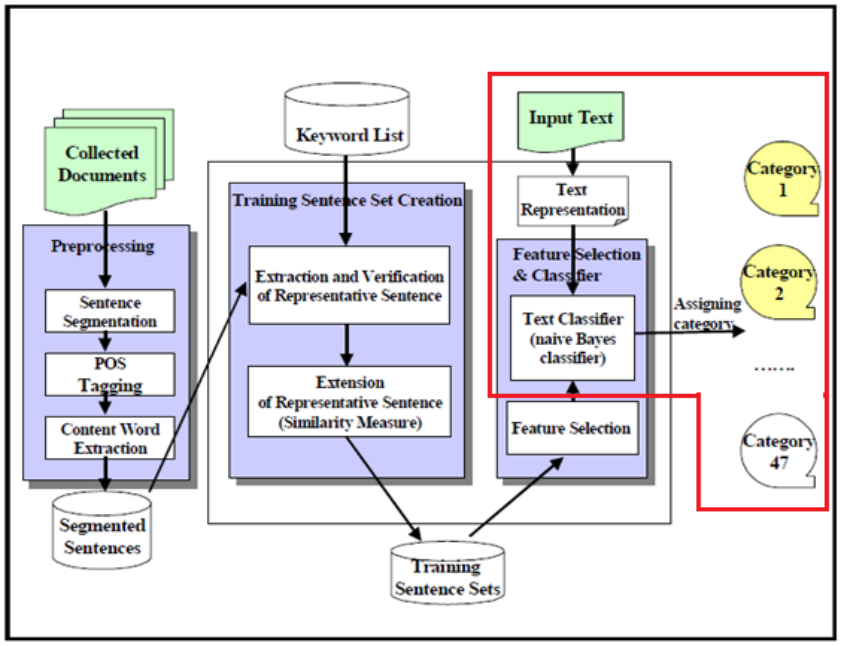


Figura2-3: Método de Etiquetado no Supervisado

2.5 Selección de atributos

Una de las etapas más importantes en clasificación es la *selección de atributos*.

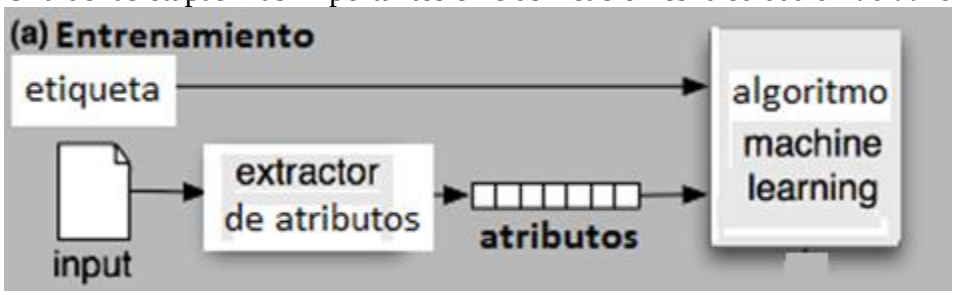


Figura2-4: Selección de Atributos

Yan y Pedersen (14) hacen un estudio comparativo de diferentes procedimientos para la selección de atributos en clasificación de textos comenzando por el más simple: umbral de frecuencia documental en el que se eliminan aquellos términos (*atributos*) que aparecen muy raras veces en los documentos de una clase. Además, es de fácil comprensión y utilización el método de la *Chi cuadrado* ($\chi 2$) que usa tablas de contingencia que muestran la frecuencia con que aparece cada atributo en cada clase etiquetada y contrasta la hipótesis de independencia entre *término* y *clase* utilizando la f.de D. $\chi 2$.

2.6 Lematización

En los lenguajes donde es frecuente el uso de formas flexionadas de un mismo *lexema* (es decir, en plural, en femenino, conjugada, etc) la *lematización* consiste en hallar el *lema* correspondiente a una palabra, es decir, la *forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra*. Hallar el ***lema*** correspondiente es equivalente a una ***entrada de diccionario***.

De esta manera múltiples representaciones y tiempos de una palabra (‘*ataco*’,’*atacante*’,’*atacado*’…) pueden reducirse a un solo *token*- ***atac***[4]- y, por tanto, a un solo atributo.

Una buena lematización requiere un conocimiento del contexto en que se usa la palabra, para lo cual es necesario realizar el *etiquetado gramatical* (*POS Tagging*) del texto. No obstante, es más frecuente, sencillo y rápido la utilización de una forma tosca de lematización llamada *stemming* que prescinde del etiquetado gramatical y que, entre otras aplicaciones, se ha utilizado en Google Search desde 2003 (15).

De los algoritmos de *stemming* de uso libre, el más conocido es el de Porter (16) que se encuentra incluido dentro de *Snowball* (17) que es un software para procesamiento de cadenas de caracteres orientado a la creación de algoritmos de *stemming*[5].

Se han llevado a cabo diferentes ensayos de evaluación de la eficacia de este tipo de algoritmos en distintos idiomas. Hollink et al. (18) manifiestan que las técnicas básicas para ciertos métodos auxiliares (palabras vacías, *stemmers*,..) deben adaptarse al idioma que se esté utilizando. Básicamente comparan la eficacia de la recuperación de información sobre textos sin *stemmizar* o *stemmizados* para siete idiomas europeos (alemán, español, finlandés, francés, holandés, inglés y sueco).

La conclusión fue que la mejora en español al usar *stemming* fue significativa (Figura2-5).

Mean average precision scores for the word-based baseline runs, the stemmed runs, and the lemmatized runs. Best scores per language are in boldface.

Language	Word-based (baseline)	Stemmed	% change	Lemmatized	% change
Dutch	0.4482	**0.4535**	+1.2%	–	
English	0.4460	**0.4639**	+4.0%	0.4003	−10.2%
Finnish	0.2545	**0.3308**	+30.0%▲	–	
French	0.4296	**0.4348**	+1.2%	0.4116	−4.2%
German	0.3886	**0.4171**	+7.3%△	0.4118	+6.0%△
Italian	0.4049	**0.4248**	+4.9%	0.4146	+2.4%
Spanish	0.4537	**0.5013**	+10.5%▲	–	
Swedish	0.3203	**0.3256**	+1.7%	–	

Figura2-5: Resultados de las Pruebas de Stemming.

2.7 Clasificadores y costes

Ya hemos manifestado las razones que hacen imprescindible la utilización de métodos supervisados de clasificación, de los cuales hemos seleccionado (§Anexo C):

1. Naïve Bayes (19)

[4] Con *SnowballStemmer*.

[5] Utilizaremos en castellano *troncalización* para la palabra *stemming*.

    a. Binomial
    b. Multinomial
    c. Gaussiano
2. Vecinos más próximos (K-NN). (20)
3. Árboles de decisión. (21)
4. Redes Neuronales. (20)

En general el desempeño de los clasificadores (equilibrados) se mide por la *tasa de error* que produce, lo que implica que los costes de los errores de clasificación - falsos negativos y *falsos* positivos - son simétricos. Sin embargo, existen casos en que el coste de un falso positivo – resultado anormal de una mamografía p.e. – es diferente de un falso negativo – resultado normal de una mamografía - son distintos. Esto suele ocurrir con más frecuencia en aquellos casos en que, como el nuestro, la clase de interés (resultado de mamografía, tuit de odio, ...) es minoritario y, como ya vimos en §2.2, una mala clasificación produce excelentes valores para la tasa de error, pero los costes de esta mala clasificación son elevados.
Este asunto se analiza con más detalle con ayuda de la curva ROC en el §Anexo D.
En nuestro caso, el prefiltrado de tuits elimina el desequilibrio de clases y la utilización de los parámetros:

- Precisión ***p*** y
- Exhaustividad ***r***

En lugar de la tasa de error, permite al usuario realizar una evaluación de clasificadores orientada por su valoración de los costes de error.
Es probable que si el usuario da más importancia a la *exhaustividad* (recuperar el máximo de elementos de la clase $C+$) que a la precisión elegirá un clasificador con mayor ***r*** y menor ***p***, lo que provoca un aumento de los falsos positivos que, por consiguiente, tienen para el usuario un coste menor que los falsos negativos.
Los diferentes ajustes que pueden hacerse tanto en el filtro como en el clasificador elegido, permiten al usuario elegir el punto (p, r) que más le satisfaga.

2.8 Herramientas informáticas disponibles

Para la ejecución del proyecto, nos interesan las herramientas para minería de datos disponibles para el analista. Tales herramientas se refieren tanto al campo del *PLN* propiamente dicho como al de los *algoritmos* de clasificación y recuperación de información que son necesarios para clasificar los tuits.
A continuación, mencionamos algunas de las herramientas y plataformas existentes que se encuentran en permanente evolución.

2.8.1 Plataformas de código abierto

- Basadas en Python:
    - Natural Language Tool Kit (NLTK) (22), para procesamiento del lenguaje natural.
    - Gensim (23),para análisis y recuperación de información textual.
    - Scikit-learn (24), biblioteca Python para análisis y minería de datos.
    - TensorFlow (25), biblioteca de código abierto para aprendizaje automático.
    - Pattern (26), bibliotecas de Python para minería en la web.
    - Keras (27), para redes neuronales.
- Basadas en Java:
    - CoreNLP (28)

- MALLET (29), programas para modelizado temático y clasificación de textos.
- LingPipe (30)
- Weka (31)
- yTextMiner (32) plataforma desarrollada en la universidad de Yonsei, Corea que integra parte de los modelos anteriores con otras bibliotecas tanto para modelización temática como clasificación de textos y análisis de sentimiento.
- Apache Lucene (33).

2.8.2 Plataformas comerciales

Probablemente la más interesante es la que forman los diferentes paquetes del Sistema *SAS™*[6], más concretamente:

- *SAS™ Enterprise Miner*: paquete de aprendizaje automático y minería de datos enfocado a *business analytics*
- *SAS™ Text Miner*: utiliza la técnica SVD (la misma que gensim) para simplificar y acelerar la clasificación de textos y extracción de información.

[6] Existe una versión - *SAS university edition* - que puede utilizarse temporalmente de manera libre (https://www.sas.com/en_us/software/university-edition.html) pero no incluye las herramientas de minería de datos.

3 Diseño

El diseño del proyecto se basa en dos partes:

- Parte I: *Investigación y diseño* de un clasificador de tuits de odio.
- Parte II: *Implementación de un detector* de tuits de odio basado en los resultados de la Parte I *utilizable por el cliente o usuario* con el que se lleva a la práctica , con diferentes métodos y herramientas, lo planificado en la fase de diseño.

3.1 Parte I

Para llevar acabar esta primera parte, se ha dividido la funcionalidad necesaria en pequeños módulos que unidos realizan las tareas precisas para esta parte:

- Obtención de tuits.
- Limpieza de tuits.
- Extracción de información de los tuits.
- Selección de tuits.
- Etiquetado de tuits.
- Selección de atributos.
- Clasificación supervisada.
- Validación de los resultados.

3.1.1 Obtención de tuits

Para recolectar la muestra de tuits que servirán como fuente de los conjuntos de entrenamiento, validación y prueba es necesario idear un plan de captura de tuits de tal forma que *nuestras peticiones siempre sean aceptadas por los servidores de Twitter*. Por ello se propone la creación de *varias cuentas* de Twitter para no exceder el número de peticiones a la API Rest de Twitter. Como requisito[7], los tuits a ser descargados tienen que *proceder de España* y *estar en español*. Cada tuit se compondrá de: identificador, autor, fecha y texto. A lo largo de las demás etapas se trabaja con un *corpus* de aproximadamente 1.000.000 de tuits obtenidos en la etapa actual.

3.1.2 Limpieza de tuits

Se deberá de tener en cuenta el formato Unicode del texto contenido en los tuits por lo que se pueden encontrar caracteres extraños en los tuits tales como: *kanjis*, eñes, tildes… Estos caracteres al igual que los *emojis* deben de ser eliminados ya que la extracción de información de los mismos es complicada. También se lleva a cabo la eliminación de los signos de puntuación: puntos, comas, puntos y comas. Todos los caracteres a ser borrados se consideran *stopwords*. Tambi*én* se realiza la eliminación de más de tres duplicados de un carácter consecutivo convirtiéndolo en solo uno, al no haber ninguna palabra en español con tres caracteres consecutivos iguales, se considera que el usuario que ha escrito el tuit muestra énfasis en lo que quiere expresar. Finalmente se transforma todo el tuit a caracteres en minúsculas.

[7] Solo nos interesan estos tuits.

3.1.3 Extracción de información de los tuits

Una vez que se han *limpiado* los tuits, se extraerán de estos los tipos de palabras relevantes en español, es decir, tipos que transmitan información acerca del contenido semántico del tuit. Para realizar esto se utiliza un *analizador morfológico*, también conocido como *POS-Tagger*. Este analizador morfológico permite diferenciar entre diferentes tipos de palabras: verbos, nombres, preposiciones, adverbios… En este caso se usan verbos, nombres y adjetivos ya que este tipo de palabras proporcionan la mayor parte de la semántica de un texto en español. El *POS-Tagger* actúa también como eliminador de palabras vacías. Además, se introducen en los tuits nuevos términos con significado que posteriormente puedan ser de utilidad. Por ejemplo: "@usuario" se sustituye por USER (se menciona a un usuario), "#tema" se sustituye por HASHTAG (indicando un tema de Twitter), y ""se sustituyen por TOKCOMI… De alguna forma estos nuevos símbolos permiten obtener información sobre: la mención a un usuario, mención de una temática o la presencia de comillas que pueden mostrar si es una cita textual o si hay cierta ironía en la palabra entrecomillada.

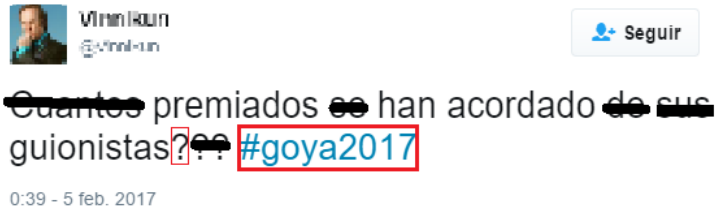


Figura 3-1: Tuit una vez se ha extraído la Información.

En la Figura 3-1se puede observar el resultado de extracción de palabras relevantes de un tuit. Las palabras eliminadas se muestran cruzadas por una línea horizontal y los términos en cajas rojas muestras términos a transformarse en símbolos que contienen significado concreto.

3.1.4 Selección de tuits para etiquetado

Como se menciona en la introducción el problema tratado se puede comparar a buscar una aguja en un pajar por lo que no se pueden seleccionar 1.000.000 de tuits con los que trabajar, esto se debe a que es probable que únicamente se encuentren 1.000 tuits de odio entre el millón y además este trabajo solo se podría realizar mediante un etiquetado manual que en tales dimensiones y proporciones es imposible llevarse a cabo. Por ello la selección de tuits se realiza mediante el uso de *un filtro* basado en términos que contengan connotación negativa y dotado de mecanismos de control del *ancho de* la *banda* de paso.

Solo pasarán por el filtro aquellos tuits que contengan términos considerados de odio por el filtro. Como se mencionó en §2.1 es este un método utilizado de un modo u otro en la detección de lenguaje de odio o insultante para equilibrar medida la proporción de tuits de odio y no odio.

El diseño del filtro está basando en conjuntos de palabras que contengan odio en distintas temáticas: raza, género, religión…cuya obtención se hace de modo iterativo partiendo de un *núcleo* inicial extraído de hatebase (34) que se va ampliando progresivamente. Sin embargo, debe tenerse en cuenta que ciertas palabras pueden resultas ambiguas en el momento de decidir si la palabra connota odio o no p.e. negro, comunista, facha… Por lo tanto, se necesita aportar una

graduación de odio a los términos y para ello se considera necesario un conjunto de palabras adicional de "*soporte al odio*". Este conjunto contiene adjetivos peyorativos y despectivos genéricos, es decir, pueden acompañar a cualquier palabra de los temas mencionados anteriormente.

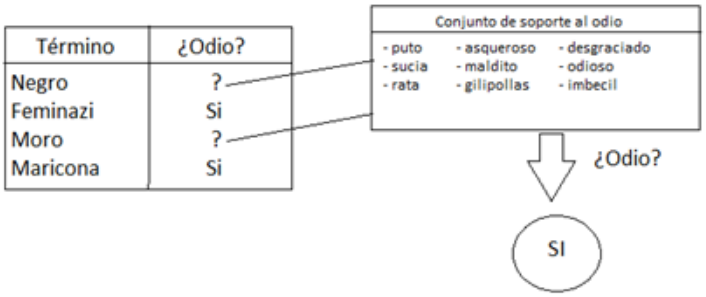


Figura 3-2: Uso de Conjunto de Soporte al Odio

En la Figura 3-2 se muestra cómo se utilizaría el conjunto de términos de soporte al odio. Claramente, términos como "*feminazi*" y "*maricona*" ambos de la categoría "sexual/género" se consideran ofensivas por sí mismas por lo que podemos decir que el tuit que contiene dichos términos puede pasar por el filtro. Sin embargo, los términos "*moro*" y "*negro*" no siempre contienen odio p.e. "hoy he tenido un día muy negro" o "los moros huyeron de la península". Por lo tanto, en ese caso es necesario el uso de un desambiguador: el conjunto de soporte al odio. Si un término de soporte al odio se encuentra en el mismo tuit que un término ambiguo respecto al odio, entonces consideramos que el tuit puede pasar el filtro.

3.1.5 Etiquetado de tuits

Durante esta etapa será necesario etiquetar *manualmente* los tuits que han pasado el filtro anterior. Estos tuits ya etiquetados serán utilizados posteriormente como conjunto de clasificación para realizar las pruebas necesarias con el objetivo de desarrollar un buen clasificador. Es necesario tener en cuenta que todos aquellos tuits que no pasen por el filtro se considerarán que pertenecen a la clase de no odio. Hay que tener en cuenta es que el etiquetado se verá influenciado por la ideología y el estado del ánimo del etiquetador ya que la interpretación de odio y no odio tiene cierto grado de subjetividad. Para resolver este tipo de problemas Para realizar el etiquetamiento correcto de los tuits es conveniente que se lleve por más de una persona obteniéndose la calificación por mayoría de votos, para lo cual se suele escoger un número impar de etiquetadores para que no haya empates.

3.1.6 Selección de atributos

Hasta este punto del proyecto ya contaríamos con un conjunto *limpio* de tuits que han *pasado el filtro* y están *correctamente etiquetados*. Sin embargo, únicamente tendríamos texto plano y este es difícil de procesar por clasificadores supervisados. Por lo que se propone representar los tuits con el modelo *BOW* (bolsa de palabras). Este modelo permite obtener características o atributos de los tuits más fácilmente. Dichos atributos aportarán la información necesaria para que el clasificador permita determinar si un tuit connota odio o no.
Se proponen las siguientes variables como atributos:

- Aparición de un término en un tuit (1 si está, 0 si no está).

- Frecuencia de un término en un tuit.
- *tf-idf*

Figura 3-3: Ejemplo de Tabla de Atributos

En la Figura 3-3 se muestra la transformación de un tuit ya limpio a su *patrón* para clasificar usando una tabla de atributos de cinco términos. En el ejemplo se utiliza el modelo binario: 1 o 0 (está o no el término en el tuit).
Las anteriores vectores están compuestos por *unigramas* (cada componente es un solo término), pero se podrían utilizar agrupaciones de n términos consecutivos (*n-gramas*). En este caso proponemos realizar pruebas con bigramas (2-gramas), para comparaciones con el uso de unigramas. El uso de bigramas y unigramas no son excluyentes por lo que ambos pueden encontrarse en el mismo vector de atributos.
Una vez se han seleccionado el modelo de atributos es conveniente utilizar un método de selección de los atributos relevantes, para reducir la dimensión del vector. Puede que la tabla de atributos que hemos generado inicialmente contenga atributos no relevantes a la hora de decidir si un tuit pertenece a la clase de odio o no odio. Para realizar esta tarea hemos optado por el test de *Chi cuadrado* ($\chi 2$) que permite contratar la dependencia entre variables (atributos) y la clase de pertenencia del tuit.

3.1.7 Clasificación supervisada

Llegados a este apartado, ya se tiene construido el conjunto de clasificación formado por *patrones*, donde cada *patrón* representa un tuit, etiquetados previamente a mano con su clase correspondiente (odio o no odio) y representado por un vector de atributos relevantes, esto es, que aportan información sobre la clase de pertenencia. Por lo tanto, se puede iniciar la fase de *clasificación* supervisada. Para ello es necesario seleccionar el tipo de particionamiento y las características del mismo. Para llevar acabo los experimentos se usa *particionamiento cruzado* y *estratificado* del conjunto de entrenamiento para mantener la proporción de *patrones* con clase de odio y no odio en cada una de las particiones. A continuación, se procedería a estimar y comparar la siguiente selección de clasificadores:

- Clasificador *vecinos próximos* o *K-NN*
- Clasificador *naïve Bayes*
- Clasificador basado en *redes neuronales*
- Clasificador basado en *árboles de decisión*

Durante la fase de desarrollo habría que tener en cuenta las diferentes variantes que tiene cada clasificador como sus métricas y los parámetros que hay que ajustar para

cada uno de ellos. Para verificar la bondad de un clasificador se tendrán en cuenta la *tasa de error* y la *desviación estándar* de la misma y se llevará a cabo un análisis de las matrices de confusión generadas. Dentro de esta misma fase se describen las comparaciones de los clasificadores.

3.1.8 Validación de los resultados

Para realizar esta etapa se seleccionan los mejores clasificadores de la fase anterior, ya se ha realizado un análisis de los resultados obtenidos con el conjunto de clasificación. También se fijan las mejores métricas, estructuras y parámetros basándonos en el análisis realizado previamente.

Para evitar el *sobreajuste* de los clasificadores se utilizarán *nuevos tuits* para realizar la validación, son tuits que no se encuentran dentro del millón utilizado para desarrollar los experimentos. Por lo tanto, es necesaria la obtención de nuevos tuits, en este caso se utilizarán 100.150 tuits aleatorios, en español y de España, es decir, capturados en condiciones similares al millón utilizado previamente para experimentos, pero en otras en otras fechas. La fase de validación permite probar los clasificadores como si de una situación real se tratara, es decir, en esta fase se determina si un clasificador es útil o no.

3.2 Parte II

En esta parte se lleva acabo el desarrollo del clasificador que será usado como herramienta por el cliente o usuario. Por lo tanto, se deben especificar *requisitos* que debe de cumplir el clasificador:

3.2.1 Requisitos funcionales

RF1: Entrenamiento del clasificador.

El usuario debe poder entrenar previamente el clasificador antes de su uso.

RF2: Clasificación de tuits.

El usuario puede clasificar un tuit introduciendo el texto contenido de este en el clasificador. El clasificador devuelve 1 si el tuit contiene odio hacia un grupo de personas o una persona en particular. En caso de no haber odio, el clasificador devuelve 0.

RF3: Clasificación de tuits.

El usuario puede seleccionar usar el prefiltrado, utilizando el mismo filtro que se usó para obtener el conjunto de clasificación, antes de realizar la clasificación del tuit deseado.

3.2.2 Requisitos no funcionales

Clase rendimiento:

RNF1: Se trata de un clasificador que funciona en modo online (en el aire) por lo que la respuesta del clasificador no puede ser muy lenta.

RNF2: El clasificador debe de tener una *exactitud superior* a la del clasificador a priori, en este caso el que indica que todos los tuits no son de odio.

Clase usabilidad:

RNF3: La utilización del clasificador debe de ser sencilla e intuitiva para el usuario.

RNF4: El clasificador contendrá un manual de usuario donde se especifica su funcionamiento.

Clase portabilidad:

RNF5: El clasificador podrá ser utilizado en sistemas que tengan Python 2.7 instalado.

3.2.3 Clasificador orientado al usuario

Para satisfacer los requisitos especificados en los dos apartados anteriores es necesario completar la primera parte del proyecto, es decir, la fase de experimentación para decidir que clasificadores se adecúan a los requisitos del usuario. Posteriormente, se realiza el encapsulamiento del clasificador en un módulo de Python que pueda ser importado fácilmente para ser su utilizado por el usuario. Aparte, también se desarrollará un *manual de usuario* para que este pueda usar el clasificador con la mayor comodidad posible. Este manual se contiene las instrucciones correspondientes tanto para el uso del clasificador como la instalación de la infraestructura software requerida (Python y las bibliotecas necesarias para el funcionamiento del sistema de clasificación).

4 Desarrollo

El objetivo de este apartado es explicar cómo se ha llevado a la práctica el diseño del proyecto descrito anteriormente y las decisiones que han ido tomándose con el objetivo de elaborar un modelo de clasificador que funcione correctamente satisfaciendo los requisitos del usuario.

4.1 Parte I

4.1.1 Obtención de tuits

Para desarrollar este punto (§3.1.1) se ha necesitado registrar 4 usuarios en la aplicación de Twitter. A partir de cada uno de los usuarios se crean 3 aplicaciones de usuarios (número máximo permitido). Por lo tanto, se han generado 12 aplicaciones en total con las que tendremos acceso a la API Rest de Twitter. La necesidad de creación de tantas aplicaciones se debe a que la API de Twitter tiene un número de peticiones limitadas cada 15 minutos. En el caso de la búsqueda de tuits el número de peticiones máximo en esta ventana de tiempo es de 180, con cada una la cuales pediremos 100 tuits (cantidad máxima por petición), por lo tanto 18.000 tuits cada 15 minutos. Al exceder esta cantidad de peticiones, la siguiente aplicación se encargará de realizar las peticiones hasta alcanzar el límite, mientras la anterior aplicación queda en espera. Por lo tanto, al realizar las peticiones con la decimoquinta aplicación, la primera ya habrá esperado lo suficiente como para volver a empezar a enviar peticiones. En la consulta de cada petición se especificará que los tuits están en español y que se obtengan los más recientes. Para asegurarnos de que los tuits se originan en España, hemos tenido que acceder a los *metadatos* de la petición para extraer el huso horario de la fecha del tuit, de tal forma que este coincida con el de España. De alguna forma forzamos a que el tuit esté en español y que tenga el huso horario de España, con ello conseguimos que sea muy probable que el tuit se haya originado en España. Finalmente, se consiguió descargar 886.532 tuits con el sistema mencionado anteriormente.

4.1.2 Limpieza de tuits

Tras haber descargado los 886.532 tuits es necesario llevar a cabo la limpieza de su contenido. El limpiador se ha desarrollado en el módulo "`limpiador.py`". La limpieza se realizará mediante el uso de *expresiones regulares* que permitirán: eliminación de espacios innecesarios, eliminación repetición de caracteres o símbolos, detección de enlaces a dominios web, detección de menciones a usuarios, detección de temas de Twitter (*hashtags*) e imposición de uso exclusivo de caracteres del alfabeto latino. Las tildes se han eliminado durante este proceso y las eñes se convertirán en enes. Finalmente, en este módulo también se llevará a cabo la creación de nuevos símbolos (*tokens*) dotados de significado.
La Tabla 4-1muestra la correspondencia de la semántica contenido del tuit a cada uno de los nuevos símbolos insertados en el texto limpiado:

Semántica	**Símbolo/Token Type**
Enlace a un dominio web	TOKENURL

Mención a un usuario	USER
Temática de Twitter	HASHTAG
Interrogación	TOKENQUES
Exclamación	TOKENEXC
Risas usando lenguaje de redes sociales: XD,xd,XDD	TOKENXD
Comillas	TOKENCOMI
Risas normales: jajaja, ajaja, jaj,…	TOKENRISAS
Incredulidad o sorpresa: WTF	TOKENWTF

Tabla 4-1: Correspondencia de Semántica y Símbolo

4.1.3 Extracción de información de los tuits

Durante esta etapa se lleva a cabo la extracción de información relevante dentro de un tuit. Se ha considerado que las palabras con más aporte de semántica para este problema de clasificación de tuits son: verbos, nombres y adjetivos. Por lo que se necesita hacer uso de un *POS-Tagger* o etiquetador morfológico. El analizador morfológico se implementará en el módulo "`POS Tagger.py`". Para desarrollar el analizador morfológico se han utilizado las bibliotecas *Pattern* y *NLTK*. También ha sido necesario descargar el *Wikicorpus* para entrenar el analizador morfológico. Este *Wikicorpus* está formado por todos los artículos de Wikipedia de 2006 y en total contiene 120 millones de palabras etiquetadas con su clase correspondiente. Para realizar el entrenamiento del *POS Tagger* ha sido necesario:

1. *Cargar todos los términos del vocabulario* contenido en los artículos del *Wikicorpus*, almacenando por cada término las diferentes clases que toma dicho término con la frecuencia correspondiente. Únicamente nos quedamos con las 100.000 palabras más frecuentes y por cada palabra su tipo más frecuente. El resultado obtenido se debe almacenar en un fichero (persistencia para su posterior uso).
2. *Generar un conjunto de reglas* para determinar a qué clase de palabra pertenece un término *basándose en su contexto*. Para ello se hará uso de la biblioteca NLTK que contiene un módulo de entrenamiento basado en el *algoritmo de Brill*. Las reglas son muy sencillas ya que se basan en el análisis de la clase de palabras que rodean el término que estamos analizando, es decir, su contexto. Algunas reglas son: verificar la clase de la palabra que se encuentra una posición adelante o dos posiciones adelante, verificar las palabras que se encuentran una posición adelante o dos posiciones adelante, verificar el tipo de palabra que hay una posición atrás, etc. El propio módulo de entrenamiento almacenará aquellas reglas que maximicen la precisión de la predicción para su posterior uso.
3. *Generar un conjunto de reglas basadas en sufijos* que permitan determinar palabras que nunca antes se han visto. Se han utilizado los cinco últimos

términos de 2.000.000 de términos del *Wikicorpus* para obtener una predicción basada en sufijos. En la siguiente Tabla 4-2 se muestran algunas relaciones entre sufijos y clases de palabras:

Frecuencia	Sufijo	Clase de palabra	Ejemplo
5.986	-ación	99% nombre común 1% preposición	natación
4.875	- mente	98% adverbio 1% preposición 1% verbo	tranquilamente
3.276	-iones	99% nombre común 1% verbo	vacaciones

Tabla 4-2: Relación de Sufijos y Clases de Palabras

Tras haber realizado la limpieza de tuits, en esta etapa tendremos 885.758, es decir, 774 tuits menos que en la etapa anterior. Este hecho puede deberse a que tras la limpieza estos tuits no contuvieran término alguno por lo que decidimos descartarlos, otros se han descartado porque la codificación no permitía la interpretación de los mismos, es posible embeber en los tuits cierto contenido en HTML que no se ha considerado útil en el proyecto, sucede lo mismo con las imágenes, vídeos o imágenes que se pueden añadir a los tuits, su contenido no se tiene en cuenta. También es de destacar que tras la limpieza habrá un total de 204.041 términos únicos (*token types*).

4.1.4 Selección de tuits

Una vez finalizada la limpieza se procede al *filtrado* (selección) de tuits a etiquetar ya que el etiquetamiento de 885.758 es inviable. Para seleccionar los tuits se desarrollará el filtro mencionado en el apartado de diseño (§3.1.4). Todo este proceso está implementado en el archivo "`kimchi.py`". Los filtros contienen las siguientes categorías de palabras fomentadoras de odio:

- *Apariencia* (`clases.txt`): hace referencia a la apariencia física y forma de ser percibida externamente p.e: marcarras, chonis y canis.
- *Discapacidad* (`discapacidad.txt`): hace referencia a las diversas discapacidades que pueden presentar las personas y anormalidades físicas que pueden ser objetivo de burla p.e: mongolo, retrasado y enano.
- *Género y orientación sexual* (`genero.txt`): hace referencia a la discriminación o desprecio por género o tendencia sexual p.e: marica, bollera y malfollada.
- *Política* (política.txt): hace referencia al insulto, menosprecio y discriminación por la condición política de la persona p.e: podemita, fascista y facha.
- *Etnicidad y nacionalidad* (`raza.txt`): hace referencia al insulto, desprecio y discriminación por la nacionalidad o etnicidad de la persona: moro, sudaca y panchito.
- *Religión* (`religión.txt`): hace referencia a las creencias religiosas de los individuos p.e.: católico y ateos.

Sin embargo, no todas las palabras expresan odio en todos los contextos, por ello se ha creado una clase nueva de palabras basada en la que se contienen las palabras de soporte al odio mencionadas en (§3.1.4). Estas palabras están contenidas en el

archivo "`maldad.txt`" y algunas de ellas son: *sucio, mierda, asqueroso* y *maldito*.
Para hacer uso del conjunto de soporte de odio en los archivos: "`clases.txt`", "`clases.txt`", "`genero.txt`", "`política.txt`", "`raza.txt`" y "`religión.txt`" se ha indicado el grado de odio contenido en cada término de los archivos. Hemos definidos dos grados: *odio absoluto* y *odio relativo*. Por un lado, el grado de odio absoluto especifica que la palabra sin ningún contexto ya transmite odio de por sí p.e: *feminazi, malfollada, guachupino*. Por otro lado, el grado de odio relativo indica que la palabra necesita estar en un contexto malicioso (el tuit contiene palabras de soporte al odio) para que esta se considere de odio absoluto p.e: *negro, retrasado* y *gitano*.
El núcleo del vocabulario (§3.1.4) procede del vocabulario en español de *hatebase* ampliado con algunos términos ausentes, pero de uso relativamente frecuente en España.
Tras desarrollar el filtro se ha procedido a su utilización y únicamente se han dejado pasar aquellos tuits que contuvieran palabras de odio absoluto o palabras de odio relativo y palabras de soporte al odio.
Una vez que se ha finalizado el proceso de selección de tuits, se consigue reducir el número de tuits a 1.971 en la primera iteración. Tras esta primera iteración se añadieron 151 palabras a los filtros y tras esta adición se realizó una segunda iteración en la que se obtuvieron 3.303 tuits, es decir, pasaron 1.332 tuits nuevos.
Hemos de mencionar que el uso del *stemming* en esta fase permite ampliar el ancho de banda del filtro, consiguiendo una mayor cantidad de tuits para etiquetado manual (Figura 4-1).

4.1.5 Etiquetado de tuits

En esta etapa ya contamos con tuits limpios y filtrados. El etiquetado no se ha podido realizar en las condiciones óptimas mencionadas en el apartado de diseño de esta categoría (§3.1.5), esto es debido a que es más rápido que una sola persona externa al desarrollo del clasificador se encargue de leer y etiquetar los tuits filtrados. Tener a tres personas leyendo y etiquetando tuits iba a ser lento y costoso ya que el etiquetado no solo se realiza una vez sino, tres:

1. Etiquetado tras la primera iteración de filtrado.
2. Etiquetado tras la segunda iteración de filtrado.
3. Etiquetado para pruebas de validación.

El tener tres personas realizando cada una de estas etapas podía suponer un retraso del proyecto al tener que leerse y etiquetar cada uno tantos tuits y posteriormente hacer la selección de la etiqueta correcta de cada tuit por votación mayoritaria.
Con una sola persona etiquetando tuits, se debe de tener en cuenta que el etiquetado es influido por la ideología de esa persona e incluso su estado de ánimo a la hora de realizar el etiquetado puede influir en el mismo. Esto es debido a que el hecho de considerar que un tuit contiene odio es muy subjetivo, salvo en algunos casos donde el odio es percibido por todo el mundo.
Durante la primera iteración del filtrado se obtuvieron 1.971 tuits de los cuales 1.362 (69%) eran de no odio y 609 (31%) eran de odio. Según los resultados anteriores, se puede observar que durante la primera iteración se consiguió un equilibrio bastante significativo de ambas clases ya que en un primer momento como se mencionó en la introducción (1) la proporción de tuits de odio antes de pasar el filtro podía aproximarse a (1:1000). Tras esta primera iteración se realizó la segunda en la que se introdujeron 151 palabras nuevas en los archivos que

constituyen el filtro. En esta segunda etapa se obtuvieron 3.303 tuits de los cuales 2404 (73%) eran de no odio y 899 de odio (27%). Se puede observar que con respecto a la primera iteración hay un leve aumente del desequilibrio entre ambas clases, en concreto una diferencia del 4%. Esto hecho puede deberse a que las nuevas palabras añadidas al filtro no contenían tanta connotación de odio como las anteriores. Finalmente, se llevará a cabo otra fase de etiquetación en validación.

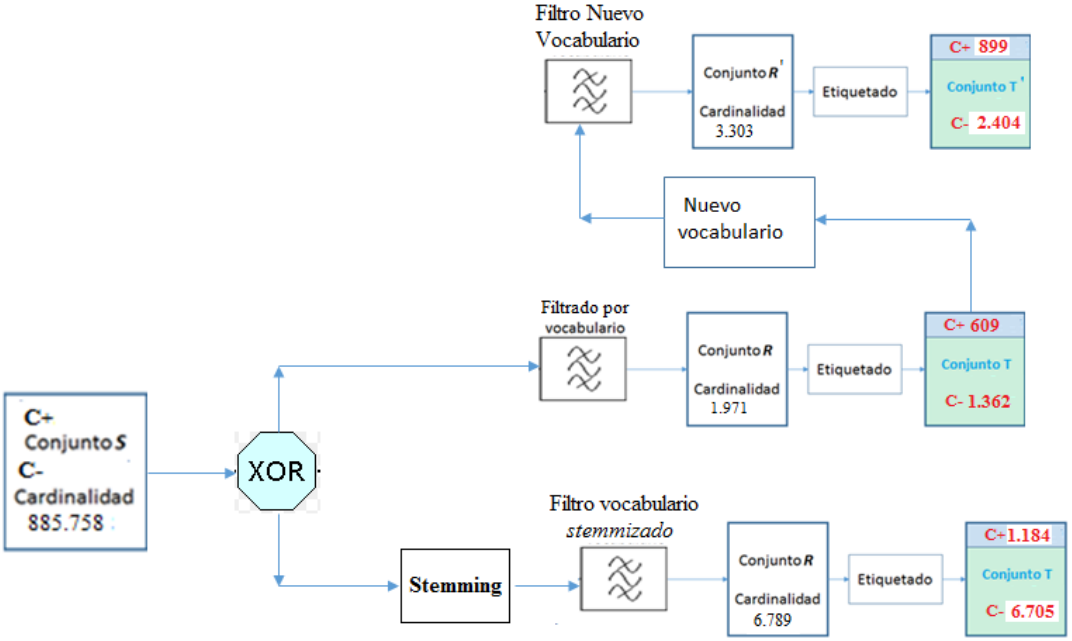


Figura 4-1: Filtrado de Tuits

El conjunto obtenido por *stemming* s no se ha utilizado en el desarrollo del proyecto, y las cifras de etiquetado son estimaciones al haberse etiquetado solo una muestra del fichero filtrado con *stemming*.

4.1.6 Selección de atributos

Durante esta etapa se realiza la extracción de atributos de los tuits siguiendo el diseño mencionado en (§3.1.6). Se utilizan como variables de atributos tanto *tf-idf* (variable continua) como la aparición o no del término en un tuit (variable binaria: 0 o 1). En el apartado de diseño se planteaba el uso de la frecuencia del término en el tuit, pero durante esta etapa de desarrollo se ha considerado no necesario tras realizar un análisis de las pruebas realizadas en (§6).

Dos variables utilizadas:

- *Binaria*: el uso del modelo *bolsa de palabras en* los tuits permite determinar fácilmente si un término se encuentra o no dentro del tuit ya que cada uno de los términos se representa con un identificador, por lo tanto, se tiene que comprobar que en el tuit ese identificador aparezca al menos una vez.
- *Tf-idf*: su utilización se debe al aprovechamiento de tuits que no pasaron el filtro. Si bien se piensa, los tuits que no pasaron el filtro (una gran mayoría) no se vuelven a utilizar una vez que se ha obtenido el conjunto de clasificación filtrado, este hecho supone una pérdida de información. Por lo que es conveniente el aprovechamiento de estos tuits para calcular el *tf-idf* de cada término en los tuits. Todo este proceso se desarrolla con *Gensim*. Se calculará el *tf-idf* sobre el *corpus* entero y se aplicará posteriormente sobre cada uno de los tuits del conjunto de clasificación ya filtrado.

En esta etapa no solo se utilizarán términos individuales (unigramas) para la representación de los tuits sino también *bigramas*. Para construir estos bigramas es necesario tener el orden relativo de las palabras dentro de cada tuit, por lo que no se puede aplicar el modelo *bolsa de palabras* (BOW) en el que el orden de los términos no es relevante. El algoritmo de construcción de bigramas es sencillo, solo hay que representar el tuit como un vector de términos en orden e ir agrupándolos de dos en dos.

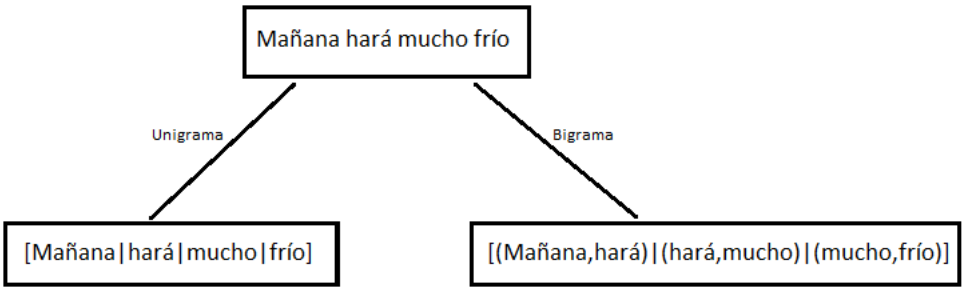


Figura 4-2: Diferencias entre Unigrama y Bigrama

En la Figura 4-2 se muestran las diferencias entre los unigramas y bigramas.
Una vez realizada la transformación a bigramas, los tuits si pueden representarse con el modelo de *bolsa de palabras*, en este caso se trataría de un *bolsa de unigramas y bigramas*, ya que

1. el uso de bigramas y unigramas no es excluyente en la tabla de atributos y
2. el orden de los términos dentro del tuit no se tienes en cuenta.

Esta estructura de bigramas también es compatible con el uso de variable binaria y *tf-idf* ya que los bigramas pueden considerarse términos. Sin embargo, durante el desarrollo por simplicidad solo se utilizan los bigramas variable binaria. Los bigramas candidatos a atributos se obtendrán a partir de la unión de palabras que constituyen el filtro junto con el resto de palabras contiguas dentro de los tuits, si se utilizaran todos los bigramas existentes dentro de los tuits que han pasado el filtro, el número de bigramas sería excesivamente grande.
La selección de atributos se lleva a cabo mediante el test de *Chi cuadrado* (15)-$\chi 2$- que se encuentra disponible en la biblioteca Scikit-learn. Se ha fijado un *p-valor* menor que 0.35 con el cual se consigue un menor error durante la clasificación supervisada. Sin embargo, se debe de tener en cuenta que para aplicar esta prueba los atributos tienen que ser discretos, como es el caso de la variable binaria. Por lo tanto, podríamos discretizar *tf-idf* pero utilizaremos la variable binaria para aplicar el test de *Chi cuadrado* ($\chi 2$). Una vez realizado el test se determina si se quiere mantener la variable binaria en representación de los atributos o si se prefiere utilizar *tf-idf*.
También se ha utilizado durante esta etapa el test de *Chi cuadrado* ($\chi 2$), para seleccionar nuevas palabras y añadirlas al filtro. Como el test devuelve aquellos atributos que son dependientes de una de las dos clases, habrá que obtener aquellos términos cuya dependencia esté asociada a la clase de odio. Para ello se han guardado aquellos términos devueltos por el test de *Chi cuadrado* ($\chi 2$) como relacionados con la clase de odio. Finalmente, hay que leer los términos obtenidos y determinar si contienen odio para añadirlos posteriormente al filtro. El procedimiento mencionado se utilizó en la segunda iteración de filtrado para

aumentar el tamaño del mismo. Se leyeron un total de 1.261 palabras de las cuales un 12% contenían odio y pudieron ser añadidas al filtro. Este procedimiento tiene la ventaja de que el usuario puede encontrar nuevas palabras sin mucho esfuerzo ya que el proceso es semi-supervisado.

El conjunto de entrenamiento se genera con el módulo "`training_set_generator.py`" y la selección de atributos se lleva a cabo en el mismo módulo de clasificación "`hate_classifier_tester.py`"

4.1.7 Clasificación supervisada

Una vez obtenido el conjunto de clasificación con los atributos relevantes seleccionados y utilizando las variables que se desean probar (binaria y *tf-idf*) se procede a la experimentación con los clasificadores mencionados en (§4.1.7). Para los clasificadores basados en redes neuronales se ha utilizado la biblioteca *Keras* que se ejecuta sobre *Tensorflow*, para los demás clasificadores se ha utilizado *Scikit-learn*. El uso de los clasificadores se realiza en el módulo "`hate_classifier_tester.py`" del proyecto.

4.1.7.1 *Árboles de decisión*

Al realizar las pruebas para estudiar el rendimiento de este clasificador se ha utilizado la clase *DecisionTreeClassifier* de *Scikit-learn* y se ha especificado que el modelo a utilizar es el basado en la entropía. Esto último se debe a que se quiere determinar cuáles son los atributos que proporcionan mayor ganancia de información respecto a las clases de odio y no odio. Al haber dos clases se formará un árbol binario a una altura h, se propone ir aumentando la altura del árbol al realizar los experimentos. Las pruebas y el análisis de las mismas se encuentran en (§5.1.1).

4.1.7.2 *Naïve Bayes*

Para desarrollar este apartado se han utilizado los clasificadores: *BernoulliNB*, *MultinomialNB* y *GaussianN*B. Estos clasificadores se encuentras disponibles en la biblioteca Scikit-learn. *BernoulliNB* se ha utilizado cuando los atributos son binarios, es decir, está o no el término en el tuit. Si además se quiere tener en cuenta la frecuencia de cada término en los tuits se utilizar *MultinomialNB*. Finalmente, se propone el uso de *GaussianNB* para variables continuas, en nuestro caso *tf-idf*, suponiendo que la distribución de los atributos sigue una distribución normal. Al igual que cuando utilizamos Naïve Bayes suponemos la independencia de atributos y en muchas ocasiones se obtienen buenos resultados, se ha creído en la posibilidad de obtener buenos resultados al suponer que la distribución de los atributos es una gaussiana aunque realmente esto no sea cierto. Las pruebas y el análisis de las mismas se encuentran en (§5.1.2).

4.1.7.3 *Vecinos Próximos (K-NN)*

Para realizar las pruebas con este clasificador se ha utilizado el clasificador proporcionado por Scikit-learn, *NeighborsClassifier*. Al realizar las pruebas se han variado el número de vecinos próximos y se han utilizado dos métricas para calcular las distancias: *Hamming* y *coseno*. La primera se ha utilizado con variables binarias, mientras que la segunda con la variable *tf-idf*. Al utilizar este clasificador se pretende averiguar si tuits que comparten términos pertenecen a la misma clase.

Además, *tf-idf* proporciona más información ya que además tiene en cuenta la frecuencia documental (idf) y la frecuencia de términos dentro del tuit (tf).
En este apartado es de destacar que K-NN con distancia coseno no estaba disponible en las bibliotecas de *Python* utilizadas por lo que ha sido necesario la implementación del clasificador con distancia coseno. Para desarrollar este clasificador propio se ha utilizado la biblioteca *Gensim*, ya que al tener el corpus con la estructura de bolsa de palabras (*BOW*) generada por la propia biblioteca, el cálculo de *tf-idf* se puede realizar con mayor facilidad utilizando Gensim. Los *tf-idfs* calculados son sobre el corpus inicial antes de utilizar el filtro, es decir, sobre los 885.758. Esto se ha realizado debido a que posteriormente no se han vuelto a utilizar los tuits no filtrados lo cual significa que hay un desperdicio de información, si se calcula el tf-idf sobre el corpus total, se cree que se podría §obtener información útil para la posterior clasificación. Además, para realizar esta clasificación no se han utilizado los atributos obtenidos por el test de *Chi cuadrado* ($\chi 2$), sino el tuit entero una vez que este está limpio, aunque solo se utilizan para clasificar los tuits del conjunto de clasificación, los 3.303, ya que estos están etiquetados y nos permiten realizar pruebas con su correspondiente análisis. El clasificador propio se encuentra en el archivo "`cosine_K-NN.py`" y su módulo de pruebas es "`cosine_tester.py`". Las pruebas y el análisis de las mismas se encuentran en (§5.1.3).

4.1.7.4 *Redes Neuronales*

Para desarrollar este apartado se ha empezado utilizando en un primer momento la biblioteca *Scikit-learn* y posteriormente se ha hecho uso de *Keras*. Esta última permite una mejor personalización de la red neuronal que se quiere diseñar, es decir, especificar mejor la estructura de la red y además proporciona una mayor cantidad de parámetros a ajustar que *Scikit-learn*, los cuáles son muy importantes a la hora de realizar una correcta clasificación.
En primer lugar, se realizaron pruebas con un perceptrón simple y cuya función de activación es *sigmoidal*. Las pruebas realizadas con este clasificador (§5.1.4.1) no mostraron grandes resultados por lo que se procedió a un cambio de estructura. Este cambio se basa principalmente en la adición de una nueva capa oculta, en este caso hay un parámetro nuevo que ajustar, el número de neuronas en esta nueva capa oculta. Las neuronas de la nueva capa oculta utilizan funciones de activación *sigmoide* al igual que las neuronas de la capa de salida. Sin embargo, los resultados obtenidos en las pruebas (§5.1.4.2) realizadas con esta nueva estructura no mostraron mejoría alguna, por ello mismo, se decidió realizar un cambio radical en la estructura de la red neuronal. Hasta este momento con las redes neuronales mencionadas anteriormente, únicamente se había utilizado la representación binaria de atributos, es decir, está o no el término. Al no haber obtenido grandes resultados anteriormente, se plantea el uso de la variable *tf-idf* normalizada. También se utilizan nuevos parámetros como el *dropout*, permitiendo así evitar el sobreajuste de la red neuronal a los datos de entrenamiento. Igualmente, se realiza un cambio en la estructura interna de la red neuronal, se utilizan tres capas ocultas en lugar de una y además las funciones de activación pasan a ser *RELU*, menos la de salida que sigue siendo *sigmoidal*, ya que queremos una probabilidad de que sea de odio, o no. Adicionalmente, se incluye en este nuevo clasificador un optimizador *Adamax* que permite no tener que ir variando la tasa de aprendizaje en las pruebas.
La nueva estructura propuesta es la siguiente:

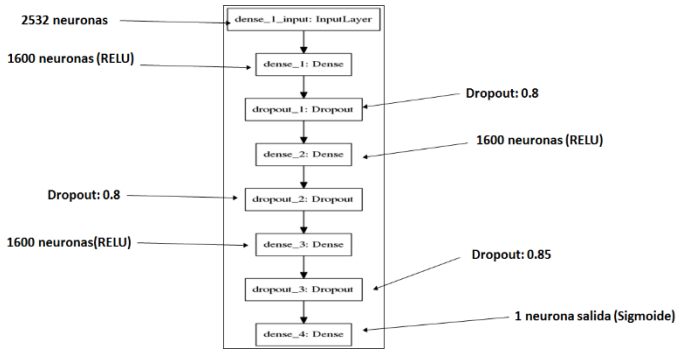


Figura 4-3: Perceptrón multicapa personalizado

Las pruebas realizadas (§5.1.4.3) con esta nueva estructura proporcionaron unos mejores resultados que los obtenidos anteriormente. El clasificador desarrollado se encuentra en el archivo "`neural_hate_classifier.py`", es un clasificador construido con la librería *Keras* pero perfectamente compatible con los clasificadores de *Scikit-learn*.

4.1.8 Validación de resultados

Tras realizar las pruebas con los diferentes clasificadores mencionados en (§4.1.7) y después de realizar el análisis de los resultados (§5.1), se llega a la conclusión que los tres mejores clasificadores son:

1- BernoulliNB utilizando unigramas.
2- BernoulliNB utilizando bigramas.
3- Perceptrón multicapa de 3 capas ocultas.

Sin embargo, no se puede afirmar que los clasificadores sean los mejores bajo condiciones reales, ya que es posible que haya habido cierto sobreajuste a la hora de seleccionar los atributos y los parámetros de cada clasificador. Por lo tanto, ha sido necesario descargar más tuits para validar los resultados de cada clasificador. En concreto se han descargado 100.151 tuits nuevos y en las mismas condiciones que los utilizados para generar el corpus, es decir, que sean en español y procedentes de España. También han sido sometidos al mismo tratamiento de limpieza y extracción de información que los que forman el corpus. La validación se llevará acabo utilizando previamente el filtro, para que las condiciones sean las mismas que durante la clasificación. Tras realizar las pruebas de validación y analizar los resultados de las mismas (§5.2) podemos determinar que el mejor clasificador de los propuestos para resolver el problema es el perceptrón multicapa de tres capas ocultas con la estructura propuesta en (§4.1.7.4). Las pruebas de validación se llevan a cabo en el módulo "`hate_validation.py`"

4.2 Parte II

Una vez que se ha determinado el mejor clasificador en la primera parte del proyecto (red neuronal multicapa de tres capas ocultas), este será encapsulado en un clasificador que será utilizado por el usuario final.

Los datos necesarios para que el clasificador tenga funcionalidad se encuentran en el directorio "`program_data.py`". Aparte también se entregará el fichero "`tagged_3000.py`" contenedor del conjunto de entrenamiento. El clasificador contendrá únicamente dos métodos: *fit* (entrenamiento) y *predic*t (clasificación). El manual de usuario del clasificador se encuentra en una wiki de Github detallando los pasos a realizar para el uso del usuario final.

5 Pruebas y resultados

Para la selección de un modelo final que sea de aplicación práctica, se siguen los siguientes pasos:

1. Aprendizaje del modelo (fase de entrenamiento)
2. Prueba del modelo (fase de prueba)
3. Selección del modelo (fase de validación).

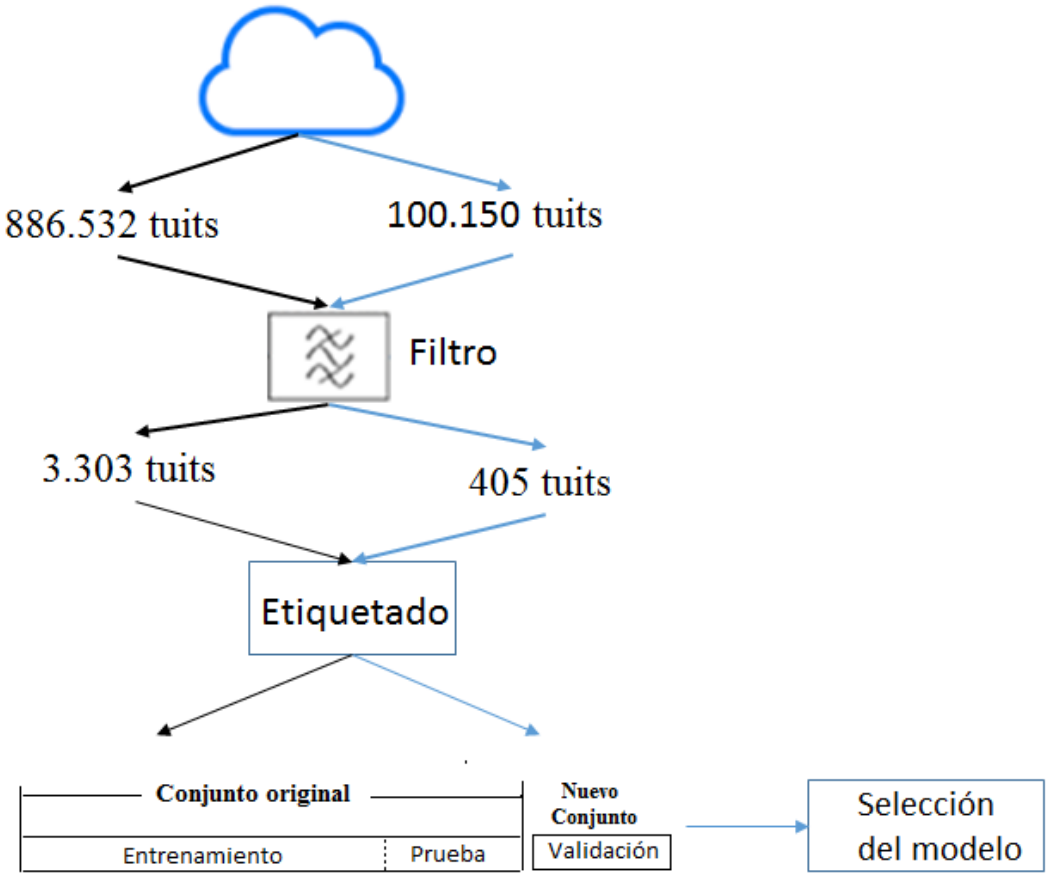


Figura 5-1: Entrenamiento y Validación de Modelos

La fase de prueba emplea la *validación cruzada*, en concreto, se hará uso de diez particiones con particionamiento *estratificado*. Esto se hace, salvo aviso en contrario, sobre el conjunto compuesto por 3.303 tuits etiquetados representado cada uno de ellos por un vector de atributos de dimensión 2.532[8]. La representación de las matrices de confusión se basará en la suma de cada partición de test (diez en total). Las demás condiciones de experimentación se especificarán en cada uno de los apartados siguientes. Cuando se especifique que hay *normalización* de los datos esta se llevará a cabo mediante utilizando *mediana* y *rango intercuartílico*(tercer y segundo cuartil).
Llamaremos ***clasificador básico*** al que clasifica todos los tuits como de no odio que tiene una *exactitud* del 72,78%.

[8] Esta dimensión puede variar según el modelo que se utilice.

5.1 Resultados de clasificación supervisada

5.1.1 Árboles de decisión

Las pruebas realizadas con árboles de decisión se realizarán con: variables binarias (1 si el término está o 0 si no está) y unigramas o bigramas. Además, el modelo de árbol de decisión utilizado es el basado en la entropía, de tal forma que el árbol se irá construyendo con aquellos atributos que proporcionen mayor ganancia de información.

5.1.1.1 Resultado utilizando unigramas.

Profundidad máxima	Tasa de error	Desviación típica entre particiones
3	0.2525	0.0121
4	0.2524	0.0146
5	0.2519	0.0101
6	0.2540	0.0139
8	0.2534	0.0161
10	0.2485	0.0133
11	0.2537	0.0195
12	0.2515	0.0165

Tabla 5-1: Comparativa de tasa de error frente a profundidad

Como se puede observar en los resultados a medida que se va aumentando la profundidad máxima del árbol la tasa de error se sigue manteniendo casi igual. Para una profundidad máxima de 12, es decir, 4.096 nodos como máximo por tratarse de un árbol binario, se ha generado la matriz de confusión de la Figura 5-2:

	Matriz de confusión	
No odio	2295	109
Odio	722	177
Real/Predicción	No odio	Odio

Figura 5-2: Matriz de Confusión de Árbol de Decisión

Como se aprecia en la matriz, aunque la *exactitud* sea casi el 75%, la *precisión* (p) y la *exhaustividad* (r) con respecto a la clase de odio toma los siguientes valores:

$$\boldsymbol{p} = \frac{177}{177+109} = 0.6188 \quad \boldsymbol{r} = \frac{177}{177+722} = 0.1968$$

Se puede observar que la *exhaustividad* es muy baja ya que se nos escapan muchos tuits que realmente sí que son de odio, la *precisión* también parce mejorable. Además, si clasificáramos todos los tuits – erróneamente - como de no

odio obtendríamos un 72,78% de exactitud por lo que 75% no se puede considerar un buen resultado.
A continuación (Figura 5-3), se muestra una instantánea de los primeros nodos del árbol de decisión (los que aportan mayor ganancia de información):

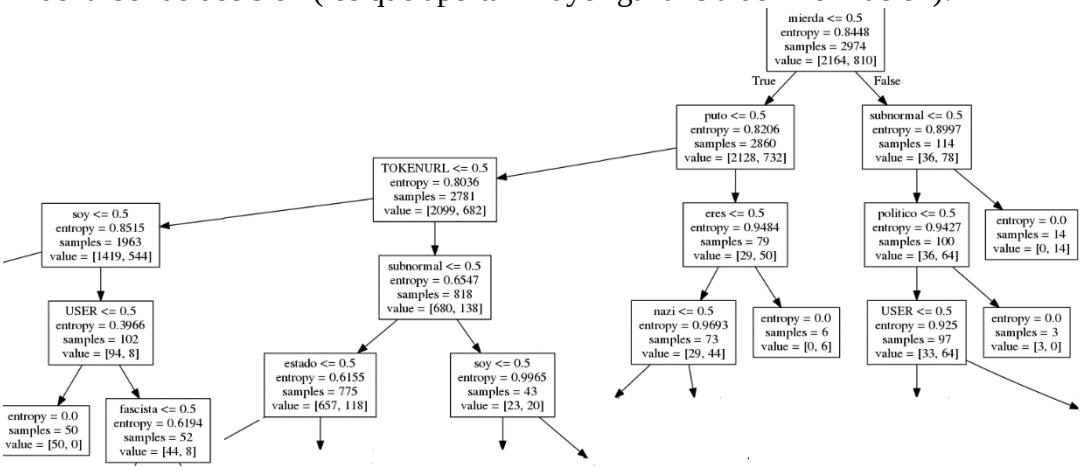


Figura 5-3: Árbol de Decisión generado

En la Figura 5-3se pueden observar algunas reglas generadas por el árbol de decisión que pueden interpretarse con sentido, p.e: si aparece la palabra "*mierda*" y "*subnormal*" se puede considerar que el tuit pertenece a la clase de odio, si aparece la palabra "*mierda*", pero no "*subnormal*", pero sí "*político*", el tuit no se considerará de odio y finalmente, si no aparece "*mierda*", si aparece "*puto*" y sí aparece "eres" el tuit es considerado de odio.

5.1.1.2 Resultado utilizando bigramas

Profundidad máxima	Tasa de error	Desviación típica entre particiones
3	0.2506	0.0118
4	0.2528	0.0126
5	0.2528	0.0108
6	0.2503	0.0085
8	0.2485	0.0130
10	0.2555	0.0113
11	0.2585	0.0313
12	0.2692	0.0215

Tabla 5-2: Comparación de exactitud con bigramas respecto a la profundidad

Se puede observar en las Tabla 5-1 y 5-2 que el uso de bigramas no aporta mucha mejoría en cuanto a *exactitud* respecto al uso de unigramas.

5.1.2 Naïve Bayes

Las pruebas realizadas con el clasificador Naïve Bayes se realizarán con:

1. variable de Bernoulli (1 el término está o 0 si no está),
2. variable de frecuencia (se tiene en cuenta el número de apariciones de un término en el tuit),
3. variable tf-idf y unigramas o bigramas.

Además, se utilizarán tres clasificadores diferentes basados en Naïve Bayes:

- Bernoulli (tiene en cuenta la aparición o no del término),
- Multinomial (tiene en cuenta la frecuencia de cada término) y
- Gaussiana.

Para el clasificador Multinomial y Naïve Bayes se ha utilizado la constante alpha (Laplace) con el valor *2.e-06* determinada empíricamente.

5.1.2.1 Resultado utilizando Bernoulli, variable binaria, unigramas

	Matriz de confusión	
No odio	2256	148
Odio	217	682
Real/Predicción	No odio	Odio

Figura 5-4: Matriz de Confusión BernoulliNB

A partir de la matriz de confusión de (8-3) se puede observar que la exactitud es de 88,94% que comparada con la del ***clasificador básico*** (predice que todos los tuits son de no odio) es superior en 16,16%, lo cual es una mejora importante respecto a los árboles de decisión, sin embargo, hay que tener otras medidas importantes como precisión (p) y exhaustividad (r):

$$\boldsymbol{p} = \frac{682}{682+148} = 0.8216 \qquad \boldsymbol{r} = \frac{682}{682+217} = 0.7586$$

La exhaustividad muestra que se nos han escapado aproximadamente un 24% de tuits que sí eran de odio, pero los clasificamos como no de odio. Mientras que la precisión muestra que el 82% de tuits que se clasificaron como de odio, realmente sí lo eran.

5.1.2.2 Resultado utilizando Multinomial, variable de frecuencia, unigramas

	Matriz de confusión	
No odio	2284	120
Odio	280	619
Real/Predicción	No odio	Odio

Figura 5-5: Matriz de Confusión MultinomialNB

La matriz de confusión (Figura 5 4) muestra que el clasificador Multinomial, que toma en cuenta la frecuencia de aparición de términos en un tuit, tiene una exactitud de 87,88%, es decir, es mayor por 15,1% con respecto al clasificador básico. A partir de la misma matriz se pueden calcular otras métricas de interés importantes como precisión (p) y exhaustividad (r):

$$\boldsymbol{p} = \frac{619}{619+120} = 0.8376 \qquad \boldsymbol{r} = \frac{619}{619+280} = 0.6885$$

Si comparamos el clasificador Bernoulli y Multinomial se puede observar que hay cierta semejanza en cuanto exactitud y precisión por lo que se puede considerar que para el cálculo de las métricas el tener en cuenta la frecuencia o únicamente si el término aparece o no, no tiene mucha importancia, este hecho podría deberse a que un tuit no contiene muchos términos lo que implica que el conteo de la aparición de términos no tenga relevancia. Sin embargo, se puede observar cierta diferencia entre la exhaustividad utilizando el clasificador Bernoulli (0.7586) y la obtenida con el Multinomial (0.6885). Esto puede deberse a que $P(x_i|y)$ al usar Bernoulli, este tiene en cuenta la probabilidad complementaria de aparición del término en la clase, mientras que Multinomial directamente lo considera como 0 si el término no aparece en el tuit a clasificar.

5.1.2.3 Resultado utilizando Bernoulli, variable binaria, bigramas

	Matriz de confusión	
No odio	2317	87
Odio	215	684
Real/Predicción	No odio	Odio

Figura 5-6: Matriz de Confusión usando bigramas y BernoulliNB

En este caso, la dimensión del vector de atributos es mayor (5.085) y la matriz de confusión (8-5) obtenida usando bigramas muestra una exactitud de 0.9085, es decir, hay menos de un 10% de tuits que se clasifican mal. Además, la precisión (p) y exhaustividad (r) son:

$$\boldsymbol{p} = \frac{684}{684+87} = 0.8871 \qquad \boldsymbol{r} = \frac{684}{684+215} = 0.7608$$

Se puede observar una mejoría con respecto al uso del clasificador Bernoulli con unigramas. El incremento de la exactitud usando bigramas es de 0.0297, el de precisión, 0.0495 y finalmente, la exhaustividad aumenta 0.0723. Este hecho tiene sentido, pues ahora podemos obtener más información acerca del contexto de los términos, información necesaria puesto que la semántica del tuit dependerá, aunque solo sea ligeramente, del contexto en el que se utilicen las palabras.

5.1.2.4 Resultado utilizando Gaussiana, variable tf-idf (normalizada), unigramas

	Matriz de confusión	
No odio	2305	99
Odio	635	264
Real/Predicción	No odio	Odio

Figura 5-7: Matriz de Confusión utilizando tf-idf y GaussianNB

La exactitud proporcionada por la matriz de confusión es de 77,78%, sin embargo, solo es mayor que la del *clasificador básico* por 4,99%. Los valores de precisión (p) y exhaustividad (r) respecto a los tuits de odio son los siguientes:

$$\boldsymbol{p} = \frac{264}{264+99} = 0.7272 \qquad \boldsymbol{r} = \frac{264}{264+635} = 0.2936$$

Se puede observar que la exhaustividad es bastante baja, hay más de un 70% de odio que no se han considerado de tal clase. El error puede deberse a que el

considerar que los atributos continuos (*tf-idf*) siguen una distribución gaussiana es incorrecto ya que los términos de un corpus suelen seguir la ley de Zipf.

5.1.3 Vecinos Próximos (K-NN)

Las pruebas realizadas con el clasificador Vecinos Próximos se realizarán con: variable binarias (1 si el término está o 0 si no está), variable tf-idf y unigramas. Las distancias utilizadas son: Hamming (variable binaria) y coseno (variable tf-idf).

5.1.3.1 Resultado utilizando K-NN distancia Hamming, variable binaria, unigramas

Número de vecinos	Tasa de error	Desviación típica entre particiones
3	0.2652	0.0074
5	0.2664	0.0058
7	0.2664	0.0045
13	0.2688	0.0018

Tabla 5-3: Tasa de Error de K-NN por Número de Vecinos (K)

Se observa en la Tabla 5-3 que las tasas de error se mantienen iguales a medida que se aumenta el número de vecinos próximos. En la matriz de confusión para k=3:

	Matriz de confusión	
No odio	2369	35
Odio	841	58
Real/Predicción	No odio	Odio

Figura 5-8: Matriz de Confusión de K-NN (distancia de Hamming)

La precisión (p) y exhaustividad (r) obtenidas de la matriz de confusión (Tabla 5-4) son las siguientes:

$$\boldsymbol{p} = \frac{58}{58+35} = 0.6236 \qquad \boldsymbol{r} = \frac{58}{58+841} = 0.0645$$

Por lo general los datos de las métricas utilizadas son muy pobres, destacado la exhaustividad que únicamente detecta el 6% de tuits que son de odio. Estos resultados tan malos pueden ser debidos a que realmente no importa que los tuits posean términos similares, también se debe de tener en cuenta el contexto que rodean a esos términos. Además, habría que añadir la cantidad de ruido que puede haber en los tuits.

5.1.3.2 Resultado utilizando K-NN distancia coseno, variable tf-idf, texto limpio del tuit original

Número de vecinos	Tasa de error	Desviación típica entre particiones

3	0.2721	0.0206
5	0.2667	0.0239
7	0.2582	0.0110
13	0.2570	0.0054

Tabla 5-4: Tasa de Error de K-NN (distancia coseno)

Como se puede observar en la Tabla 5-4, esta es muy similar a la obtenida utilizando la distancia de Hamming. En este caso también se ha realizado una matriz de confusión para k=3:

	Matriz de confusión	
No odio	2124	280
Odio	619	280
Real/Predicción	No odio	Odio

Figura 5-9: Matriz de Confusión K-NN (distancia coseno)

La precisión (p) y exhaustividad (r) obtenidas de la matriz de confusión (Figura 5-9) son las siguientes:

$$\boldsymbol{p} = \frac{280}{280+280} = 0.5 \qquad \boldsymbol{r} = \frac{280}{280+619} = 0.3114$$

La exhaustividad es bastante mayor en este caso en comparación con los resultados obtenidos con la distancia de Hamming (6,45%), sin embargo, la precisión es menor con distancia coseno, una disminución de 0.12. A pesar de que con la distancia coseno haya cierta mejoría con respecto al uso de K-NN con distancia Hamming, los resultados no se pueden considerar buenos. A pesar de que ahora se tenga en cuenta el tf e idf y la distancia coseno nos permite obtener los k tuits más parecidos, esto no significa que los tuits pertenezcan a la misma clase. Puede deberse al ruido o incluso por el propio contexto, es decir, que este último no sea vea lo suficientemente reflejado en la distancia coseno.

5.1.4 Redes Neuronales

Las pruebas realizadas con los clasificadores basados en redes neuronales se realizarán con:

1. variables binarias (1 el término está o 0 si no está),
2. variable tf-idf y unigramas.

Los tres modelos de redes neuronales utilizados son: perceptrón simple, perceptrón de una capa oculta y perceptrón multicapa de tres capas ocultas.

5.1.4.1 Perceptrón Simple

Para realizar las pruebas correspondientes con este clasificador se han tenido en cuenta dos parámetros muy importantes que son:

- el número de épocas y
- la tasa de aprendizaje.

Para calcular la tasa de aprendizaje óptima se ha establecido un número máximo de iteraciones (1.000) y esta se ha ido variando. Al realizar el experimento se obtuvo la siguiente gráfica:

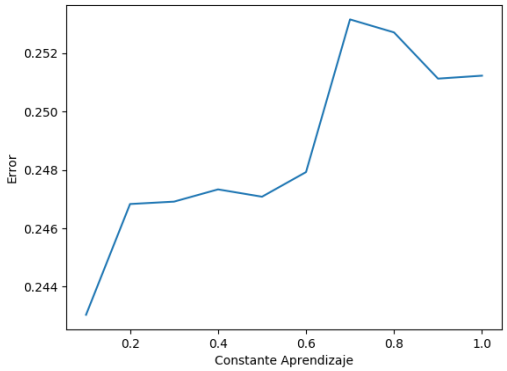


Figura 5-10: Covariación Tasa de Error-Constante de Aprendizaje

Esta gráfica muestra que la tasa de error mínima se encuentra con una tasa de aprendizaje de 0,1, es decir, el valor más pequeño con el que se ha probado. Utilizando esa tasa de aprendizaje se ha obtenido la siguiente matriz de confusión de una de las particiones:

	Matriz de confusión	
No odio	2314	90
Odio	622	277
Real/Predicción	No odio	Odio

Figura 5-11: Matriz de Confusión Red Neuronal Monocapa

La tasa de error obtenido es 21,55% si se compara con la del clasificador básico (27,21%) solo se rebajaría el error un 6%. La precisión (p) y exhaustividad (r) obtenidas de la matriz de confusión (-) son las siguientes:

$$\boldsymbol{p} = \frac{277}{277+90} = 0.7547 \qquad \boldsymbol{r} = \frac{277}{277+622} = 0.3081$$

Estos valores de la métrica pueden deberse a que se trata de un problema que no es linealmente separable por lo que un perceptrón no es capaz de clasificar correctamente los tuits en sus correspondientes clases.

5.1.4.2 Perceptrón de una capa oculta

Al utilizar una red de una sola capa oculta hay que ajustar el número de neuronas en la capa oculta aparte de la tasa de aprendizaje. Para ello se utiliza una búsqueda codiciosa de parámetros, se fija la tasa de aprendizaje mientras se varía el número de neuronas en la capa oculta y una vez encontrado el número que minimice el tasa de error se fija ese número de neuronas y se varía la tasa de aprendizaje hasta encontrar la mínima:

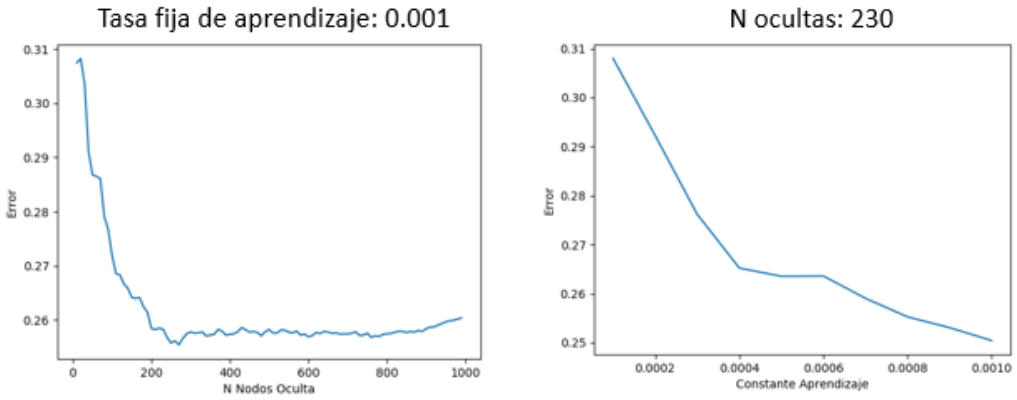


Figura 5-12: Red Neuronal con una Capa Oculta: Error vs. Tasa de Aprendizaje

Se puede observar en las gráficas que la tasa de error mínima se consigue con 230 neuronas en la capa oculta y una tasa de aprendizaje de 0,001. Clasificando con estos parámetros y fijando el número máximo de iteraciones a 1.000 se ha generado la matriz de confusión:

	Matriz de confusión	
No odio	2362	42
Odio	717	182
Real/Predicción	No odio	Odio

Figura 5-13: Matriz de Confusión de Red Neuronal con una Capa Oculta

Como se puede observarse, la tasa de error es de 22,97% mientras que la precisión (p) y la exhaustividad toman los valores:

$$\boldsymbol{p} = \frac{182}{182+42} = 0.8125 \qquad \boldsymbol{r} = \frac{182}{182+717} = 0.2024$$

La precisión ha aumentado respecto al perceptrón simple mientras que la exhaustividad ha disminuido. Puede deberse a que una única capa no sea suficiente para resolver el problema al que nos enfrentamos.

5.1.4.3 Perceptrón multicapa de tres capas ocultas

Finalmente se experimenta con un perceptrón de tres capas ocultas con 1600 neuronas en cada una obteniendo la siguiente matriz de confusión:

	Matriz de confusión	
No odio	2291	113
Odio	361	538
Real/Predicción	No odio	Odio

Figura 5-14: Matriz de Confusión. Red Neuronal de tres Capas Ocultas

Claramente se puede observar que la exactitud se ha incrementado con respecto a los dos clasificadores basados en redes neuronales examinados anteriormente, ya que ahora la tasa de error es de 14,35%, mientras que la precisión (p) y la exhaustividad (r) toman los siguientes valores:

$$\boldsymbol{p} = \frac{538}{538+113} = 0.8264 \qquad \boldsymbol{r} = \frac{538}{538+361} = 0.5984$$

La precisión se ha mantenido muy similar a los casos anteriores, aunque haya aumentado, sin embargo, se puede apreciar un apreciable aumento en la exhaustividad, es decir, se nos escapan menos tuits de odio. Esta mejoría del clasificador puede deberse a los nuevos parámetros mencionados en la estructura de esta red neuronal y de la variación de la misma con respecto a los anteriores experimentos.

5.2 Resultados de validación

Las pruebas de validación se realizan con 100.151 tuits y aplicando previamente el filtro construido (Figura 5-1: Entrenamiento y Validación de Modelos).

5.2.1 BernoulliNB (Naïve Bayes)

5.2.1.1 Variables binarias y unigramas.

Los resultados obtenidos son los siguientes:

	Matriz de confusión	
No odio	296	45
Odio	32	32
Real/Predicción	No odio	Odio

Figura 5-15: Matriz de Confusión en Validación. Bernoulli

La matriz de confusión muestra una tasa de error de 19,01%, mientras que los valores de precisión (p) y exhaustividad (r) son los siguientes:

$$\boldsymbol{p} = \frac{32}{32+45} = 0.4155 \qquad \boldsymbol{r} = \frac{32}{32+32} = 0.5$$

Estos resultados son bastante peores que a los obtenidos en validación bajo las mismas condiciones (§5.1.2). Esto podría deberse a la aparición de nuevos términos en los tuits de prueba que no están en la tabla de atributos pero, sobre todo, al *sobreajuste* de la probabilidad a priori ya que en nuestro conjunto de entrenamiento y validación el 73% de los tuits son de no odio y el 27% son de odio, mientras que el conjunto de prueba, 84% de tuits son de no odio y un 16% de odio. Como Naïve Bayes tiene en cuenta la probabilidad a priori, al no mantenerse la misma proporción de tuits en ambos conjuntos hay un aumento de error.

5.2.1.2 Variables binarias y bigramas

Los resultados obtenidos son los siguientes:

	Matriz de confusión	
No odio	296	45
Odio	35	29
Real/Predicción	No odio	Odio

Figura 5-16: Matriz de Confusión en Validación. Bernoull. Bigramas

En este caso la matriz de confusión muestra una tasa de error de 19,75% y las métricas precisión (p) y exhaustividad (r) toman los siguientes valores:

$$\boldsymbol{p} = \frac{29}{29+45} = 0.3918 \qquad \boldsymbol{r} = \frac{29}{29+35} = 0.4531$$

Al usar bigramas se observa un empeoramiento en todas las métricas. Este hecho puede deberse a que ha habido un *sobreajuste* a la hora de realizar bigramas, si antes era probable que un término de la tabla de atributos no se encontrara en el tuit a clasificar, ahora es más probable que un bigrama (compuesto por dos términos) pueda no estar en la tabla de atributos. Aunque en (§5.1.2) se hayan obtenido buenos resultados usando bigramas, a la hora de validar, los resultados son incluso algo peores que usando unigramas. También en este caso habría que tener en cuenta la diferente de proporción de tuits de odio y no odio en el conjunto de entrenamiento y el de validación, exactamente como en el caso mostrado anteriormente.

5.2.2 Validación con redes neuronales

Para realizar esta prueba de validación se utilizará la red multicapa con la estructura detallada en (§5.1.4.3). Se utilizarán variable tf-idf, normalización basa en mediada y rango intercuartílico. Los resultados de validación han sido los siguientes:

	Matriz de confusión	
No odio	333	8
Odio	25	39
Real/Predicción	No odio	Odio

Figura 5-17: Matriz de Confusión en Validación. Red Neuronal Multicapa

Se observa en la matriz de confusión que la tasa de error es de 8,14%, mientras que la precisión (p) y exhaustividad (r):

$$\boldsymbol{p} = \frac{39}{39+8} = 0.8297 \qquad \boldsymbol{r} = \frac{39}{39+25} = 0.6093$$

Los resultados obtenidos son apreciablemente mejores que los hallados anteriormente con Naïve Bayes, sin embargo, la exhaustividad es aún baja.
Esta mejora en los resultados puede ser debida a que el perceptrón multicapa no origina un sobreajuste tan severo como el mencionado para Naïve Bayes. Esto tiene sentido puesto que durante el entrenamiento de la red neuronal no aparece de forma explícita la probabilidad a priori de cada clase.

5.3 Clasificador seleccionado

Los resultados de las pruebas señalan la *red neuronal multicapa* como clasificador que presenta mejor desempeño cuyas características resumimos a continuación:

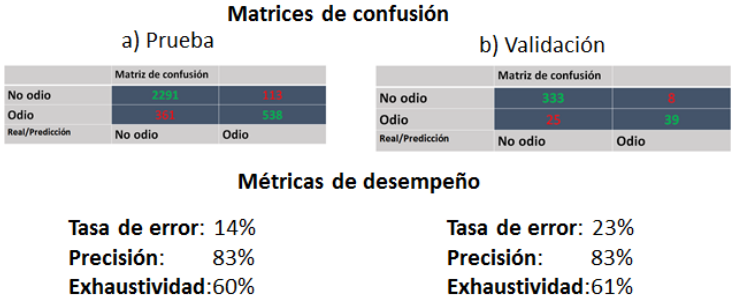


Figura 5-18: Métrica dela Red Neuronal Multicapa

6 Conclusiones y trabajos futuros.

6.1 Conclusiones

Tras haber realizado el presente trabajo de investigación orientado al desarrollo de un modelo que permita diferenciar tuits con contenido de odio y no odio, se pueden extraer algunas conclusiones:

- El problema al que hemos hecho frente, como muchos otros en *pruebas médicas*, *detección de fraude*, …, se caracteriza por la existencia de una gran diferencia entre la cardinalidad - desequilibrio - de ambas clases (problema de la *aguja en el pajar*).
- En estas situaciones, es casi siempre necesario llevar a cabo un *sobremuestreo* o submuestreo, que, en la mayoría de los casos examinados en la literatura, se realiza ya sea *explícitamente* (10) o *implícitamente seleccionando las fuentes* a analizar (9). Además, si el sobremuestreo se realiza mediante filtrado, al igual que se filtran los mensajes para los conjuntos de entrenamiento y prueba, es necesario aplicar el mismo filtro antes de realizar la clasificación supervisada ya que, si no, ambos conjuntos (filtrado y no filtrado) no seguirían la misma distribución (§Anexo B).
- Como se indica en dicho Anexo, la utilización del filtro provoca que la *exhaustividad* de la clasificación sea reducida si consideramos el conjunto total de tuits y no solo los que han pasado el filtro, caso en que los tuits que no pasan el filtro son clasificados directamente como no de odio, lo cual implica que existe una gran dependencia de los resultados de clasificación y la bondad del *filtro*.
- Este es un problema relativo para aplicaciones como la detección de tuits de odio en que tanto o más importante que clasificar el máximo número de tuits es identificar las fuentes de mensajes de odio, lo que resulta menos afectado: se pierde exhaustividad en mensajes, pero este problema *no* afecta tanto a *las fuentes*.
- Sin embargo, si se considera únicamente el conjunto de *tuits filtrados* como tuits de interés a analizar, la clasificación supervisada produce buenos resultados.
- Este tipo de proyectos requieren el uso de herramientas de *Procesamiento del Lenguaje Natural* que, para el idioma español, son escasas comparadas con el inglés, motivo por el cual algunos procedimientos que utilizan ambos idiomas dejan bastante que desear lo que nos ha obligado a entrenar un *POS-Tagger* en español que solo funciona bajo Linux.
- Respecto a la utilización del *stemming* que parece dar buenos resultados en español (18), nuestros experimentos muestran que tiene una cierta utilidad en la concepción del filtro previo al etiquetado ya que, de algún modo, amplía el ancho de banda del vocabulario ya que al *stemmizar* los términos, se incluyen también las palabras *flexionadas* de un mismo lema (plurales, género masculino/femenino, …). Sin embargo, para este problema, hemos probado la clasificación con términos stemmizados sin obtener buenos resultados (§Anexo F).
- La única clase de clasificadores utilizable es la de *clasificadores supervisados*, aun cuando los no supervisados pueden ser útiles para la reducción de la dimensión del problema - componentes principales- o para utilizar el análisis de conglomerados para facilitar el sobre o submuestreo.
- Respecto al mejor clasificador, como muestran los resultados de *validación* (§ 5.2), es la red neuronal multicapa de tres capas ocultas el que mejores resultados proporciona. Aunque Naïve Bayes mostrara los mejores resultados durante la fase de prueba (realizada, recordemos, con instancias de la misma muestra que el conjunto de entrenamiento) (§5.1), ello se debe al sobreajuste debido a la fijación de

probabilidades a priori durante el entrenamiento que son distintas de las de la muestra de validación, lo que provocó un menor rendimiento durante validación.

- El punto anterior, ilustra lo que se ha dicho más arriba (2), (4):
    - Por una parte, la frecuencia de mensajes de odio es muy variable y está ligada a eventos detonadores (2) y el *desequilibrio entre clases varía* con esta frecuencia.
    - Por otra, el *vocabulario* usado – afecta al filtro - cambia con dichos eventos.

  Todo lo cual nos lleva a la conclusión de que resulta necesario un trabajo sistemático de modificación del filtro y etiquetado de nuevos tuits.

6.2 Trabajos futuros.

Tras la finalización y análisis del proyecto, se nos plantean posibles adiciones y de funcionalidad y mejoras de las ya existentes.

Como *mejoras*, puede considerarse un futuro uso de base de datos relacionales como las SQL para poder almacenar los diferentes términos, tanto de odio como de soporte al odio, que constituyen el filtro. Durante este proyecto estos términos se almacenaron en documentos de textos. Sin embargo, debemos de ser conscientes de que si se añaden más categorías de delitos de odio y más términos - lo que se produce constantemente-, resulta más fácil la manipulación de los mismos con una base de datos.

Además, también pueden analizarse otras técnicas para hacer frente al problema de desequilibrio de clases. Una de ella podría ser dar diferentes pesos a las clases de odio y no odio.

Respecto a los clasificadores, podría ampliarse la gama de los utilizados incluyendo otros tales como: bosques aleatorios (*random forest*), *máquinas de soporte vectorial* (VSM) y otras estructuras de redes neuronales. Al tratarse de un problema de clasificación también se podría utilizar un *clasificador por votos* (Ensembler) el cual esté constituido por más de un clasificador, cada uno de los cuales aporta su voto determinando si un tuit contiene odio o no hacia un colectivo o persona concreta, siendo la predicción la clase que más votos haya obtenido, puede que este método supusiese la reducción de error en las métricas.

Respecto al *etiquetado,* en un futuro, este podría llevarse a cabo con más de un etiquetador, para evitar que el clasificador se vea influenciado por la ideología o estado de humor del etiquetador.

Además de mejoras también se proponen algunas *ampliaciones de funcionalidad* como la captura de la *autoría del tuit* de odio lo que permitiría identificar la fuente del tuit, ya sea para castigarlo o tenerlo de alguna forma vigilado. No solo eso, también se podría utilizar el nombre de usuario de *Twitter* como un *parámetro más en la clasificación*, pues un tuit con autoría atribuida a un usuario que normalmente comete delitos, podría hacer más probable el contenido de odio. En general, se podría aprovechar más la *información del usuario* como su biografía, últimos tuits generados, número de seguidores y usuarios seguidos. Estos dos últimos datos podrían utilizarse para generar grafos en dónde cada usuario generador de tuits de odio es un nodo y las relaciones, aristas, representan las relaciones existentes entre ellos.

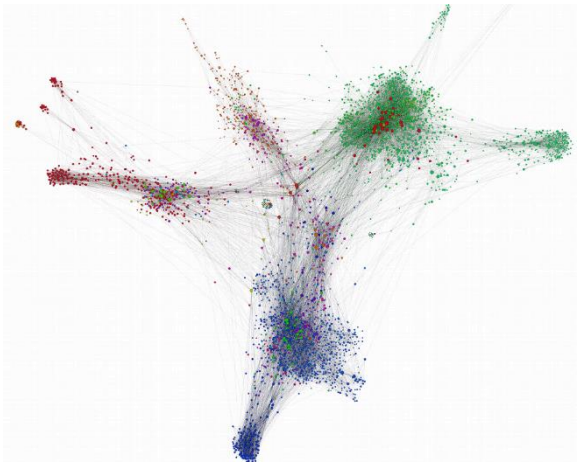


Figura 6-1: Grafo de Relaciones entre Usuarios de Twitter

A partir de los tuits de odio y su autoría se podría obtener un grafo similar al mostrado en la imagen de tal forma que mediante el uso de métricas basadas en análisis de redes sociales se puede obtener información sobre el foco principal de odio y las relaciones que mantienen los emisores de tuits de odio y sus vecinos.

Referencias

1. BOE. [En línea] 30 de Marzo de 2015. [Citado el: 22 de junio de 2017.] https://www.boe.es/boe/dias/2015/03/31/pdfs/BOE-A-2015-3439.pdf.
2. *Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making.* Pete Burnap, Matthew L. Williams. 223-242, 2015, Policy & Internet, Vol. 7.
3. *An introduction to ROC analysis.* Fawcett, Tom. s.l. : Elsevier, 2005.
4. *Up and Down with Ecology - The Issue Attention Cycle.* Downs, A. 1972, Public Interest (28), págs. 28-50.
5. *The class imbalance problem in pattern classification and learning.* V. García, et al. 2007.
6. *Análisis de Datos Multivariantes.* Peña, Daniel. s.l. : McGraw Hill, 2002.
7. Wikicorpus v.1.0. [En línea] [Citado el: 21 de junio de 2017.] http://www.cs.upc.edu/~nlp/wikicorpus/.
8. *Detecting Offensive Language in Social Media to Ptotect Adolescent Online Safety.* al., Ying Chen et. 2011.
9. *Detecting Hate Speech on the World Wide Web.* William Warner, Julia Hirschberg. Montréal, Canada : s.n., 2012. Proceedings of the 2012 Workshop on Language in Social Media.
10. Tomas Borovicka, Marcel Jirina Jr., Pavel Kordik y Marcel Jirina. Selecting Representative Data Sets. [En línea] 2012. [Citado el: 21 de mayo de 2017.] http://dx.dio.org/10.5772/50787.
11. *A review of instance selection methods.* Olivera López, J. Arturo et al. 2010, Artif Intell Rev (2010) 34, págs. 133–143.
12. *Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure.* Marco Saerens, Patrice Latinne, Christine Decaestecker. 2002, Neural computation 14(1), págs. 21-41.
13. *Automatic Text Categorization by Unsupervised Learning.* Youngjoong Ko y Jungyun Seo. 1997.
14. *A Comparative Study on Feature Selection in Text Categorization.* Yiming Yang, Jan O Pedersen. 1997. ICML '97 Proceedings of the Fourteenth International Conference on Machine Learning.
15. MIT Libraries. [En línea] [Citado el: 12 de marzo de 2017.] http://libguides.mit.edu/c.php?g=176061&p=1159432.
16. *An algorithm for suffix stripping.* Porter, M. F. 1980, Program , Vol. 14 Issue: 3, págs. 130-137.
17. Snowball. [En línea] 11 de marzo de marzo. [Citado el: 7 de abril de 2017.] http://snowball.tartarus.org/.
18. *Monolingual Document Retrieval for European Languages.* Hollink, V., Kamps, J., Monz, C. et al. 4, 2004, Information Retrieval, Vol. 7.
19. *A Comparison of Event Models for Naive Bayes Text Classification.* Andrew McCallum, Kamal Nigam. Pittsburgh : s.n., 1998.
20. Bishop, Christopher. *Pattern recognition aND Machine Learning.* s.l. : Springer, 2006.
21. Norvig, Peter y Russell, Stuart. *Artificial Intelligene. A Moder Approach.* s.l. : Prentice Hall, 2003.
22. Natural Language Toolkit. [En línea] [Citado el: 6 de junio de 2017.] http://www.nltk.org/.

23. *Software Framework for Topic Modelling with Large Corpora.* Řehůřek, Radim and Petr Sojka. Valetta, Malta : s.n., 2010. Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks. págs. 46--50.
24. *Scikit-learn: Machine Learning in Python.* Gael Varoquax et al. 2011, Journal of Machine Learning Research, págs. 2825-2830.
25. Google Research. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. [En línea] 2015. [Citado el: 21 de abril de 2017.] http://download.tensorflow.org/paper/whitepaper2015.pdf.
26. Pattern. [En línea] [Citado el: 13 de mayo de 2017.] https://pypi.python.org/pypi/Pattern.
27. Keras. [En línea] [Citado el: 17 de junio de 2017.] https://keras.io/.
28. Christopher D. Manning et al. The Stanford CoreNLP Natural Language Processing Toolkit. [En línea] [Citado el: 6 de marzo de 2017.] http://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf.
29. McCallum, Andrew. MALLET: A Machine Learning for Language Toolkit. [En línea] [Citado el: 26 de marzo de 2017.] https://people.cs.umass.edu/~mccallum/mallet/.
30. *LingPipe for 99.99% Recall of Gene Mentions.* Carpenter, Bob. Valencia,Spain : s.n., 2007. Proceedings of the 2nd BioCreative workshop.
31. *Weka: Practical Machine Learning Tools and Techniques.* Ian H. Witten et al. 2007. Proceedings of the ICONIP/ANZIIS/ANNES'99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems. págs. 192-196.
32. yTextMiner. [En línea] [Citado el: 4 de marzo de 2017.] http://informatics.yonsei.ac.kr/yTextMiner.
33. https://lucene.apache.org/core/. [En línea] [Citado el: 21 de abril de 2017.]
34. *Hatebase: An anti-genocide app.* Brown, Jesse. Toronto : Rogers Media, 2013, MacLean's.

Glosario

A

AI. *Véase* Inteligencia Artificial

alpha
: En suavizado de Laplace, es la cantidad que se usa para modificar la estimación de parámetros en variables categóricas multinomiales

análisis exploratorio de datos
: Proceso al que se someten los datos antes de su modelización. Suele consistir en resumir sus características principales, con frecuencia usando métodos gráficos. Su objetivo es permitir una planificación más adecuada tanto del proceso de recolección como de su tratamiento posterior

API
: (Application Program Interface). conjunto de subrutinas, funciones y procedimientos o métodos que ofrece una bilioteca para ser utilizados por otros programas como una capa de abstracción

aprendizaje estadístico
: Rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. Se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos

Aprendizaje Máquina. *Véase* Aprendizaje Estadístico, *Véase* Aprendizaje Estadístico

B

bolsa de palabras
: (Bag of Words)
: método que se utiliza en el procesado del lenguaje para representar documentos ignorando el orden de las palabras. En este modelo, cada documento parece una bolsa que contiene algunas palabras

BOW. *Véase* bolsa de palabras, *Véase* bolsa de palaba

business analytics
: Conjunto de habilidades, técnicas y prácticas para la exploración iterativa del desempeño pasado de una empresa a fin de obtener una mejor comprensión de su funcionamiento ysu desarrollo futuro. Utiliza tanto la minería de datos como la inteligencia artificial

C

Ciencia de los Datos
: Campo interdisciplinario que comprende los procesos y sistemas para extraer conocimiento ,,de grandes volúmenes de datos en sus diferentes formas (estructurados o no estructurados) y formatos (.txt, .dat, .doc, .jpg, etcétera

clasificación de textos
: Tareaconsistente en asignar un documento a una categoría determinada

clasificación no supervisada
: La clasificación no supervisads es aquella en la que el algoritmo clasificador no necesita de más información que el conjunto a clasificar y algunos parámetros que límiten el número de clases

clasificación *supervisada*
: Se parte de un conjunto de clases conocido a priori. Estas clases deben caracterizarse en función del conjunto de variables mediante la medición de las mismas en individuos cuya pertenencia a una de las clases no presente dudas

conjunto de prueba
: Es el usado para evaluar la bondad d las predicciones del modelo

corpus
: Conjunto amplio y estructurado de ejemplos reales de uso de la lengua. Estos ejemplos suelen ser frecuentemente textos
: EnNLP, colección de documentos

coseno
: (distancia o similitud)
: medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos

cuartil
: cuartiles son los tres valores que dividen un conjunto de datos ordenados en cuatro partes porcentualmente iguales

D

delitos de odio
: Aquellos motivados por prejuicios respecto a la víctima del mismo y tienen lugar cuando el perpetrador del delito elige a la víctima en base a su pertenencia a un cierto grupo

detonador
: Suceso que desencadena una serie de actitudes, sentimientos y acciones en el público de modo que aumenta la relevancia del tema con que se relaciona, 3

E

emojis
: ideogramas o caracteres usados en mensajes electrónicos y sitios web

entidad

Producto, persona, evento, organización o tópico, 1

error tipo I
: En contraste de hipótesis es la probabilidad de aceptar la hipótesis alternativa siendo cierta la que se contrasta

error tipo II
: En contraste de hipótesis estadísticas es la probabilidad de elegir la hipótesis que se contrasta siendo cierta la alternativa

especificidad
: Proporción de falsos negativos sobre el total de instancias negativas que proporciona un clasificdor

estado del arte
: Una de las primeras etapas dentro de un proyecto es la construcción de su estado del arte, ya que permite determinar la forma como ha sido tratado el tema, cómo se encuentra el avance de su conocimiento en el momento de realizar una investigación y cuáles son las tendencias existentes, en ese momento, para el desarrollo de proyectosen el mismo campo

etiquetado
: Asignar una etiqueta (clase) a cualquier dato del conjuntoada

Etiquetado
: Operación por la cual se asigna una 'etiqueta' a cada instancia indiando la clase a que pesta pertenece

etiquetado gramatical
: Proceso de asignar a cada una de las palabras de un texto su categoría gramatical

etnicidad
: etnicidad
  Una etnia es un conjunto de personas que tienen en común rasgos culturales, idioma, religión, vestimenta, nexos históricos, tipo de alimentación, y, muchas veces, un territorio. Dichas comunidades, a veces, reclaman para sí una estructura política y el dominio de un territorio

Exactitud
: Medida de desempeño de un clasificador. Es el porcentaje de instancias que se clasifican en su clase real

F

f.de D
: Función de distribución de una varaiable a leatoria

G

Github
: Plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git

grupo de control
: Grupo al que no se aplica el factor que se prueba en diseño de experimentos

grupo experimental
: Grupo al que se aplica el factor que se prueba en diseño de experimentos

H

Hamming
: (distancia de)
  Número de bits que tienen que cambiarse para transformar una palabra de código válida en otra palabra de código válida

hashtags
: cadena de caracteres formada por una o varias palabras concatenadas y precedidas por una almohadilla (#). Es una etiqueta de metadatos precedida de un carácter especial con el fin de que tanto el sistema como el usuario la identifiquen de forma rápida

hatebase
: Base de datos multilingüe de palabras de odio

I

identidad de género
: Percepción subjetiva que un individuo tiene sobre sí mismo en cuanto a sentirse hombre, mujer, o de un género no binario, sin considerar características físicas o biológicas

instancias
: Una instancia es cada uno de los datos de los que se disponen para hacer un análisis

Inteligencia Artificial
: *ciencia de hacer máquinas que actúan racionalmente. • Racional es todo agente que busca alcanzar unos objetivos de manera tal que optimiza el valor de una función de utilidad*

L

lema
: Forma única con la que se nombra una palabra

lexema
: Parte que se mantiene invariable en todas las palabras de una misma familia

Ley de Moore
: La ley de Moore expresa que aproximadamente cada dos años se duplica el número de transistores en un microprocesador

ley de Zipf
: Ley empírica que muestra la relación entre la frecuencia de una palabra y su rango. En efecto, mientras mayor sea el rango de una palabra, menor será la frecuencia con la que aparece en el texto

M

matriz de confusión
: Herramienta que permite la visualización del desempeño de un algoritmo que se emplea en *aprendizaje supervisado*. Cada columna de la matriz representa el número de predicciones

de cada clase, mientras que cada fila representa las instancias en la clase real

mediana
valor de una variable que deja a cada lado el 50% de los valores que toma

metadatos
datos que describen otros datos. Un grupo de metadatos se refiere a un grupo de datos que describen el contenido informativo de un objeto

métrica
Conjunto de patrones de medida mediante los cuales se evalúa el rendimiento o la calidad de un clasificador

microblogs
Servicio que permite a sus usuarios enviar y publicar mensajes breves, generalmente solo de texto

modelizado temático (*topic model*) Modelo de aprendizaje sin supervisión que permite detectar el tema o asunto del que trata un documento

N

n-gramas
secuencia de n términos dentro de una frase o texto. La probabilidad de una palabra se hace depender de las N – 1 palabras anteriores

NLP. *Véase* Procesado de Lenguaje Natural

O

orientación
Tipo de opinión respecto a una entidad favorable, desfavorable o inexistente

orientación sexual
Patrón de atracción sexual, erótica, emocional o amorosa a determinado grupo de personas definidas por su sexo

P

POS Tagger
Analizador morfológico que permite diferenciar entre diferentes tipos de palabras verbos, nombres, preposiciones, adverbios…

POS Tagging. *Véase* etiquetado gramatical

Procesamiento del Lenguaje Natural
Campo de las ciencias de la computación, inteligencia artificial y lingüística que estudia las interacciones entre las computadoras y el lenguaje humano

proporción de falsos positivos
Fracción de las instancias negativas que se clasifican como positivas

p-valor
en contraste de hipótesis, probabilidad de obtener un resultado al menos tan extremo como el que realmente se ha obtenido (valor del estadístico calculado), suponiendo que la hipótesis nula es cierta

R

rango intercuartílico
Diferencia entre el tercer y el primer cuartil de una v.a

Rest
Representational State Transfer
conjunto de principios de arquitectura para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes (como p.e. SOAP)

S

SAS
Acrónimo de Statistical Analysis Systems
Lenguaje de programación desarrollado por SAS Institute a finales de los años sesenta

sensibilidad. *Véase* exhaustividad

sobreajuste
efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado. Durante la fase de sobreajuste el éxito al responder las muestras de entrenamiento sigue incrementándose mientras que su actuación con muestras nuevas va empeorando

stemming
Forma básica de lematización que prescinde del contexto

SVD
*D*escomposición en *V*alores *S*ingulares de la matriz término-documento que permite simplificar las tareas de minería de datos

T

tasa de aprendizaje
parámero utilizado en redes neuronales que determina la amplitud del paso entre iteraciones

Tasa de error
Medida de desempeño de un clasificador. Es el porcentaje de instancias que se clasifican en una clase equivocada

tasa de prevalencia
Número de personas que padecen de una enfermedad determinada en un punto determinado de tiempo por cada 1.000 habitantes

tf-idf
(Term frequency– Inverse document frequency), frecuencia de término – frecuencia inversa de documento (o sea, la frecuencia de ocurrencia del término en la colección de documentos), es una medida numérica que expresa cuán relevante es una palabra para un documento en un corpus

token
Secuencia de letras entre delimitadores

U

umbral de corte
Puntuación de una instancia que actúa como frontera entre dos clases. Las instancias con puntuación superior al umbral se clasifican en una clase y los de puntaje inferior en otra

W

Wikicorpus
Corpus etiquetado con 120 millones de palabras en español

Anexos

Anexo A: Medida del Rendimiento de un Clasificador y Conjuntos Desequilibrados

Conjuntos de datos

Para el desarrollo de un clasificador partimos de un conjunto de instancias (*mensajes o textos*) que forman un *corpus*. Lo ideal es recoger *varios conjuntos de datos independientes*, si ello no es posible, debemos conformarnos con un solo conjunto de datos que habremos de dividir en dos o tres subconjuntos.

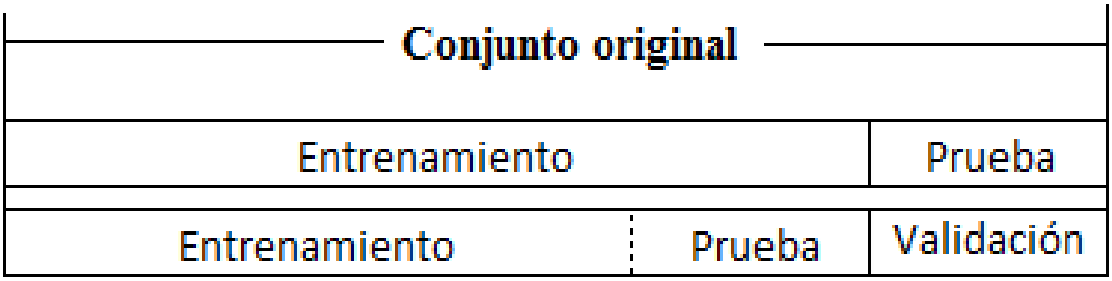


Figura 0-1: División del Conjunto de Datos

La estrategia consiste en usar

1. un *conjunto de* ***entrenamiento*** para aprender y estimar los parámetros del modelo;
2. un *conjunto de* ***prueba*** para evaluar modelos y seleccionar uno de ellos y
3. un *conjunto de* ***validación*** para valorar la capacidad de predicción de los modelos.

Existen múltiples métodos de división de datos, de los cuales el más simple es el *método de retención* (*holdout*) que consiste en dividir aleatoriamente el conjunto original en dos subconjuntos ($^2/_3$ o $^1/_2$ para entrenamiento y el resto para prueba). Si el conjunto original no es lo bastante grande, el método es ineficiente.

En *aprendizaje estadístico* se utiliza como supuesto básico que *tanto el conjunto de entrenamiento como el de prueba se extraen de una misma distribución subyacente* constituida por la combinación de las distribuciones de la clase mayoritaria y minoritaria. Si se llevan a cabo modificaciones en el conjunto de entrenamiento para tratar de reequilibrar las clases, el *conjunto de entrenamiento y el de prueba tendrán distribuciones diferentes* violando este supuesto.

Para la evaluación de clasificadores es preciso utilizar alguna *métrica* que nos permita estimar su *rendimiento*.

La *matriz de confusión*, elemento básico de evaluación de clasificadores, se expresa en el caso de que solo existan dos clases como:

		Predicción			Métricas	
		+	-		ROC	Usual
Real	+	Verdadero Positivo (TP)	Falso Negativo (FN)	π	$tp=TP/\pi$	$p=TP/Y$
	-	Falso Positivo (FP)	Verdadero Negativo (TN)	ν	$fp=FP/\nu$	$r=TP/Y$
		Y	N	M		

Tabla 0-1: Matriz de Confusión

Las medidas más inmediatas de evaluación del modelo que se nos ocurren son:

- *Exactitud*: $\boldsymbol{Acc} = \frac{TP+TN}{TP+TN+FP+FN}$
- *Tasa de error*:$\boldsymbol{Err} = \frac{FP+FN}{TP+TN+FP+FN} = 1 - \boldsymbol{Acc}$

Sin embargo, cuando *existe un claro desequilibrio entre clases*, como es el caso de los tuits de odio que pueden ser un 2‰ del total, es posible obtener una *exactitud enorme* aun clasificando todos los tuits (erróneamente) como negativos (no de odio), y *equivocándonos*, por tanto, *en todos los positivos*:

		Predicción		
		odio	*Neutro*	
Real	*odio*	0	200	200
	Neutro	200	99.800	100.000
		200	100.000	**100.000**

Tabla 0-2: Matriz de Confusión con Desequilibrio de Clases

Lo que nos daría unos valores de exactitud del 99,8%:

Acc	99,8%
Err	0,2%

Por ello, son más útiles los indicadores:

- ***Precisión*** (*p*recision) **p:** es el porcentaje de los *tuits* realmente pertenecientes a una clase que se asignan a la misma(aciertos) sobre el total de los asignados a dicha clase por el clasificador:

$$\boldsymbol{p} = \frac{TP}{TP + FP} = \frac{TP}{Y}$$

  Es decir, el porcentaje de predicciones que se acierta.

- ***Exhaustividad*** (*recall* o proporción de verdaderos positivos **TP)**: **r** es el porcentaje de los *tuits* que han sido clasificados como pertenecientes una clase sobre el total de miembros de dicha clase en el conjunto (porcentaje de instancias de la clase bien clasificadas):

$$\boldsymbol{r} = \frac{TP}{TP + FN} = \frac{TP}{\pi}$$

Se utilizan también combinaciones de p y r, tales como la media geométrica:

$$\boldsymbol{G} = \sqrt{p \cdot r}$$

y la media armónica:

$$\boldsymbol{F} = \frac{\mathbf{2}}{\frac{\mathbf{1}}{\boldsymbol{p}} + \frac{\mathbf{1}}{\boldsymbol{r}}} = \frac{\mathbf{2}\boldsymbol{pr}}{\boldsymbol{p} + \boldsymbol{r}} \qquad \textbf{Ec.(0-1)}$$

Un valor p=1 nos dice que todos los elementos recuperados como relevantes lo son, pero no nos dice nada acerca de si hemos recuperado todos los documentos relevantes.
Una cuestión importante es cómo *equilibrar los falsos positivos y los falsos negativos,* cuestión que, evidentemente, depende del *coste de las consecuencias de cada error*.
Una herramienta frecuentemente utilizada en este contexto es el ***ROC*** (3)**,** habitual en Medicina y Biología donde las consecuencias de la detección de falsos positivos y negativos puede tener costes muy diferentes.

Un *gráfico ROC* es una técnica para visualizar, organizar y seleccionar clasificadores basados en su desempeño. Su nombre se deriva de su primera aplicación a la detección de señales campo desde el que se ha venido extendiendo su uso a la medicina y a los problemas de clasificación en general.
Ahora a la *exhaustividad* citada en §1.4. -$\frac{TP}{\pi}$- se la denomina ***sensibilidad***. Como se ve, es la

$$\mathrm{P}(predicción\_\boldsymbol{TRUE}|\boldsymbol{TRUE})$$

y se introduce la ***especificidad****:*

$$\frac{TN}{TN + FP} = \frac{TN}{\nu}$$

De manera que

$$1 - especificidad = \frac{FP}{TN + FP} = \frac{FP}{\nu}$$

Y es la *proporción de falsos positivos* ***fp****:*

$$\mathrm{P}(predicción\_\boldsymbol{TRUE}|\boldsymbol{FALSE})$$

Por el teorema de la probabilidad total sabemos que:

$$\mathrm{P}(predicción\_T|T)\,P(T) + \mathrm{P}(predicción\_T|F)P(F) = \mathbf{P(predicción\_T)} =$$
$$= \boldsymbol{tp} \cdot P(T) + \boldsymbol{fp} \cdot P(F) = \boldsymbol{tp} \cdot P(T) + \boldsymbol{fp} \cdot [1 - P(T)] =$$
$$= \boldsymbol{tp} \cdot \frac{\pi}{\pi + \nu} + \boldsymbol{fp} \cdot \frac{\nu}{\pi + \nu} \qquad \textbf{Ec. (0 1)}$$

Si dibujamos el gráfico que relaciona ambas magnitudes, obtenemos la Figura 0-2 en que **AUC** es el *área bajo la curva* que puede tomar valores entre 0 (no acierta nunca) y 1 (la predicción acierta siempre).

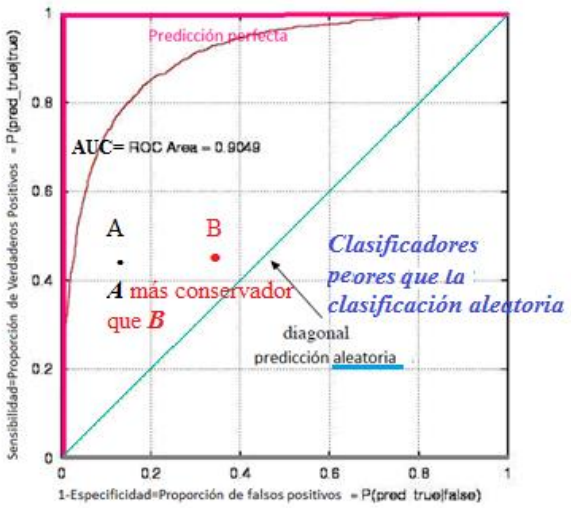


Figura 0-2: ROC

La Tabla 0-3 pone de manifiesto que mientras que los indicadores usuales (*p*, *r*) usan valores de ambas filas de la matriz, los de ROC usan *valores de la misma fila*.
Por tanto, en el primer caso cuando cambian las proporciones de las clases estas medidas cambian, aunque no lo haga el rendimiento del clasificador, cosa que no sucede en ROC como se ve en la Tabla 0-3, al cambiar la proporción de clases.

π/ν	3											
		Predicción				Métricas				Predicción				Métricas		
		+	-			ROC	Usual			+	-			ROC	Usual	
Real	+	100	20	120	*tp*	*83%*	77%	*p*	Real	+	150	30	180	*tp*	*83%*	71%	*p*
	-	30	10	40	*fp*	*75%*	83%	*r*		-	60	20	80	*fp*	*75%*	83%	*r*
		130	*30*	160							*210*	*50*	260				

Tabla 0-3Métrica ROC frente a Métrica Convencional

Anexo B: Clasificación tras Filtrado y Exhaustividad

Como hemos dicho, debido a la dificultad de etiquetado manual de un conjunto de tan gran tamaño y tan baja proporción de tuits de odio, hemos utilizado un procedimiento que filtra el conjunto inicial extraído de Twitter mediante el uso de un vocabulario con términos de odio obtenidos de diferentes fuentes y el conjunto filtrado, de una cardinalidad mucho menor, se *etiqueta* y usa como conjunto de *entrenamiento.* Seguidamente, el conjunto de entrenamiento se usa para estimar los parámetros del clasificador y, con este, se clasifican los tuits del conjunto de test.

Prefiltrado de tuits para clasificación.

Para la utilización del clasificador en otros conjuntos de tuits, caben dos opciones:

1. *Aplicarlo tras filtrar* el nuevo conjunto, asignando a todos los tuits que no pasan el filtro a *C-*.
2. *Aplicarlo al conjunto completo* con lo cual algunos de los tuits que no pasaron el filtro, se clasifiquen en *C+* (correcta o incorrectamente).

En el primer caso – el seguido por nosotros- (Tabla 0-4), la *matriz de confusión* cuando se ***aplica a la totalidad del conjunto inicial***, muestra como FN a las instancias que no pasan el filtro que debieran estar etiquetadas con (+), una en nuestro ejemplo y, evidentemente, estas instancias que no pasan el filtro

		Predicción				
		+	-			
Real	+	3	2	***5***	**Precisión (p)**	75,0%
	-	1	17	***18***	**Exhaustividad (r)**	60,0%
		4	***19***	**23**	**F**	66,7%

Tabla 0-4: Matriz de Confusión del Conjunto Inicial

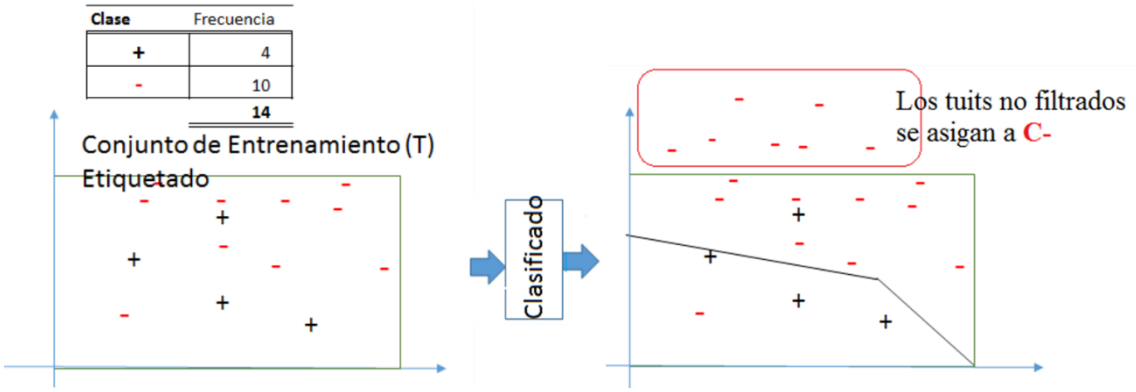


Figura 0-3: Clasificación tras Filtrado

Medidas de desempeño de clasificadores.

Ya se ha mencionado (§1.4) que la *matriz de confusión* permite medir el rendimiento de un clasificador y se expresa en el caso de que solo existan dos clases como:

		Predicción	
		Positivo	***Negativo***
Real	***Positivo***	Positivo Verdadero (**TP**)	Falso Negativo (**FN**)
	Negativo	Falso Positivo (**FP**)	Negativo Verdadero (**TN**)

Tabla 0-5: matriz de confusión.

Exhaustividad del método.

Al proceder a la clasificación de tuits tras su filtrado, *inevitablemente, se produce una exhaustividad muy baja* como veremos a continuación.

Supongamos que disponemos de un conjunto de 100.000 instancias en el cual hay 1.200 instancias *C+* y 88.000 *C-*. Se aprecia claramente el desequilibrio de clases (1,2%) frente a 98,8 %).

Sea que hemos seguido el proceso de filtrado para la obtención del conjunto de entrenamiento:

1. Para el ***etiquetado***, se parte del conjunto $\mathcal{S}$ (muestra) de tamaño 100.000 que *se filtra* mediante una lista de vocabulario obteniendo un nuevo conjunto $\mathcal{R}$ de mucha menor cardinalidad (1.000) que *se etiqueta manualmente*, produciendo el conjunto de entrenamiento $\mathcal{T}$ de tamaño 1.000, a partir del cual se procederá a la estimación de los modelos de clasificación como muestra la Figura 0-4.

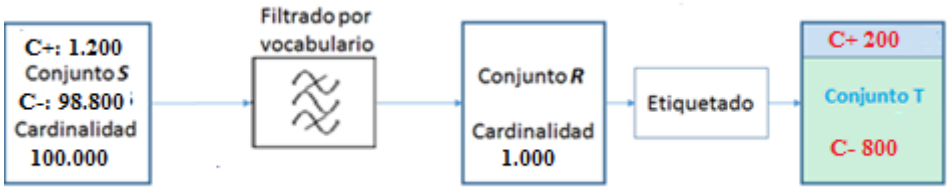


Figura 0-4: Prefiltrado de Tuits

   Seguidamente se procede a la prueba de los modelos (Bayes, K-NN, Redes Neuronales, …) a fin de seleccionar el más adecuado.

2. Para la ***clasificación***, se opera también en dos fases:
   a. *Filtrado* del conjunto inicial;
   b. *Clasificación* del conjunto filtrado.

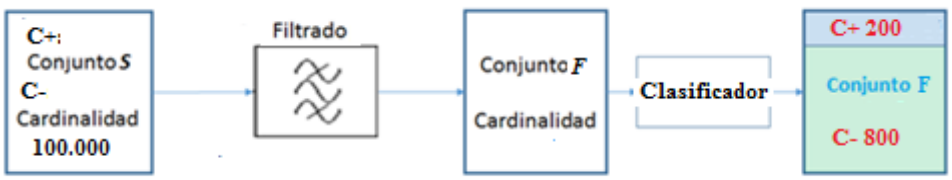


Figura 0-5: Clasificación tras Filtrado

Este procedimiento presenta un serio e inevitable problema: su *exhaustividad* es siempre muy pequeña, como se ve en el siguiente ejemplo:

1. ***Matriz de confusión del conjunto F***:

		Predicción +	Predicción -	
Real	+	180	20	200
	-	10	790	800
		190	810	1.000

Figura 0-6: Matriz de Confusión de Conjunto Prefiltrado

   - Precisión: 95%
   - Exhaustividad (*respecto al conjunto F*): 90%

2. **Matriz de confusión en el conjunto *S***: En este caso, los casos que no pasan el filtro se clasifican como C+, por lo cual:

		Predicción -
Real	+	1.000
	-	98.000
		99.000

		Predicción +	Predicción -	
Real	+	180	1.020	1.200
	-	10	98.790	98.800
		190	99.810	100.000

Figura 0-7: Matriz de Confusión con Prefiltrado para el conjunto S

Y se obtiene:

   - La misma precisión: 95%
   - Exhaustividad (*sobre conjunto S*): 15%

Con lo cual, el método tiene ***baja exhaustividad***.

Clasificación sin prefiltrado.

Cabe pensar en una clasificación sobre el conjunto sin filtrar

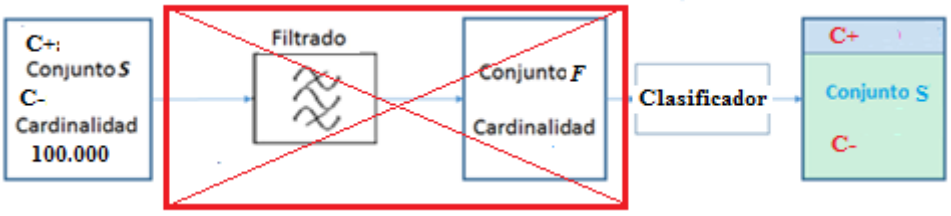


Figura 0-8: Clasificación sin Prefiltrado

Sin embargo, es evidente que, al aumentarse el ruido en el conjunto de entrada, la precisión debe reducirse y, a cambio, aumentará la exhaustividad, como p.e. según los resultados siguientes:

		Predicción		
		+	-	
Real	+	800	400	1.200
	-	500	98.300	98.800
		1.300	98.700	100.000

Figura 0-9: Matriz de Confusión sin Prefiltrado previo

De donde se obtiene:

- Precisión: 61%
- Exhaustividad: 66%

Anexo C: Clasificadores

Naïve Bayes

Clasificador basado en el teorema de Bayes suponiendo la independencia de los atributos. Cada tuit *j* está representado por un vector de atributos (términos) $\vec{F_j}$ y conocidas tanto las probabilidades a priori $P(C_i)$ como las condicionadas $P(\vec{F_j}|C_i)$ podemos calcular para cada tuit *j* la probabilidad condicionada a su contenido de que pertenezca a la clase

$$P(C_i|\vec{F_j}) = \frac{P(\vec{F_j}|C_i)P(C_i)}{\sum_k P(\vec{F_j})P(C_k)}$$

Bernoulli.

$\vec{F_j}$: presencia/ausencia de cada término *t* en documento *j* de la clase *i*

$$P(\vec{F_j}|C_i) = \prod_{t=1}^{n}[f_{jt}P(w_t|C_i)+(1-f_{jt})(1-P(w_t|C_i)]$$

$P(w_t|C_i)$**:** probabilidad de que el término $w_t$esté en la clase C_i
n: número de atributos.

$$f_{jt} = \begin{cases} \mathbf{1} \text{ si } w_t \text{ en tuit j} \\ \mathbf{0} \text{ si } w_t \text{ no en tuit j} \end{cases}$$

Multinomial

$\vec{F}$: frecuencia de cada palabra

$$P(\vec{F_j}|C_i) = \prod_{k=1}^{n}\frac{N_{yk}+\alpha}{N_y+\alpha n}$$

T: conjunto de entrenamiento
N_{yi}**:** número de veces que i-ésimo término aparece en la clase y en T**.**
N_y**:** suma de frecuencias de todos los términos de la clase y
n: número de atributos.
α**:** constante suavizadora de Laplace.

Gaussiano

Suponemos que los atributos siguen una distribución normal.

$$P(\vec{F_j}|C_i) = \prod_{k=1}^{n}\frac{1}{\sqrt[2]{2\pi{\sigma_{ky}}^2}}{}^{-\frac{(x_k-\mu_{ky})^2}{2(\sigma_{ky})^2}}$$

σ_{ky}**:** desviación estándar de los valores del k-ésimo atributo de la clase y
μ_{ky}**:** media de los valores de los valores del k-ésimo atributo de la clase y

K vecinos más próximos (K-NN)

Clasificador basado en la situación de la instancia en el espacio y las k instancias más cercanas que le rodean. La cercanidad a las k instancias más cercanas se puede calcular con algunas de las siguientes métricas:

- Distancia euclídea: $\sqrt[2]{\sum_{k=1}^{n}(x_k - y_k)^2}$
- Distancia Manhattan: $\sum_{k=1}^{n}|x_k - y_k|$
- Distancia Hamming: $\frac{\sum_{k=1}^{n} x_k \neq y_k}{n}$

A parte de las distancias mencionadas anteriormente, para calcular la distancia entre pares de documentos es común utilizar el coseno del ángulo formado por ambos:

$$\boldsymbol{tf}(\boldsymbol{t},\boldsymbol{d}) = \begin{cases} \mathbf{1} + \boldsymbol{log_2}\big(\boldsymbol{frec}(\boldsymbol{t},\boldsymbol{d})\big) \; si \; frec(t,d) > 0 \\ \mathbf{0} \; en \; otro \; caso \end{cases} \qquad \boldsymbol{idf}(\boldsymbol{t},\boldsymbol{D}) = \mathbf{log}\,\frac{|\boldsymbol{D}|}{|\boldsymbol{D_t}|}$$

$$\boldsymbol{tfidf}(\boldsymbol{t},\boldsymbol{d}) = \boldsymbol{tf}(\boldsymbol{t},\boldsymbol{d})\boldsymbol{idf}(\boldsymbol{t},\boldsymbol{D})$$

$\boldsymbol{t}$**:** término
$\boldsymbol{d}$**:** documento (tuit)
$\boldsymbol{frec}(\boldsymbol{t},\boldsymbol{d})$**:** número de apariciones de $\boldsymbol{t}$ en $\boldsymbol{d}$
$|\boldsymbol{D}|$**:** número de documentos
$|\boldsymbol{D_t}|$: número de documentos en los que aparece $\boldsymbol{t}$

$$\cos\big(\overrightarrow{d_1},\overrightarrow{d_2}\big) = \frac{\sum_t d_{1t} d_{2t}}{\sqrt[2]{\sum_t {d_{1t}}^2 \sum_t {d_{2t}}^2}} = \frac{\sum_t tfidf(t,d_1)tfidf(t,d_2)}{\sqrt[2]{\sum_t tfidf(t,d_1)^2 \sum_t tfidf(t,d_2)^2}}$$

Cuanto mayor sea el coseno, menor será el ángulo entre los dos tuits, lo que implica una mayor semejanza.

- 1-NN: una instancia se clasifica en la clase a la que pertenece la instancia más próxima.

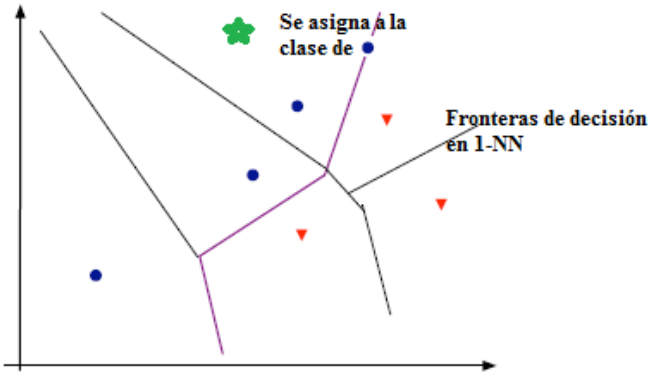


Figura 0-10: 1-NN

- k-NN: una instancia se clasifica en la clase mayoritaria de las k instancias más próximas.

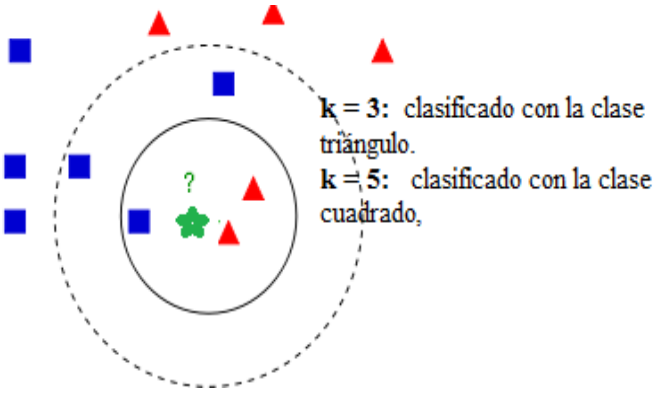


Figura 0-11: K-NN

Árboles de decisión.

Modelo de predicción basado en la toma de decisiones siguiendo un diagrama previamente construido. El valor predicho dependerá de los valores de los atributos de la instancia que se desea clasificar.
Árbol que representa un *diagrama de flujo*.

- Nodo -->test
- Rama -->valores posibles del nodo padre
- Nodos hojas -->etiquetas de clase
- Medida de atributos --> entropía/impureza de Gini

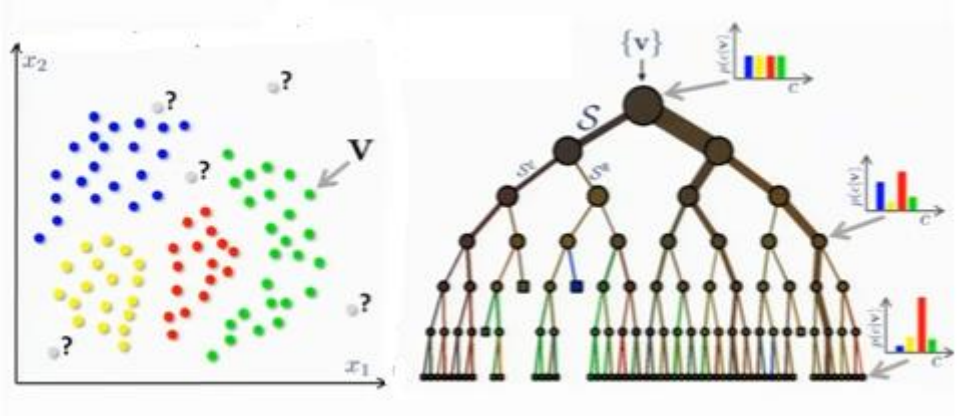


Figura 0-12: Árbol de Decisión

En cada paso de la construcción del árbol se suelen utilizar métricas para elegir el atributo que mejor discrimina la instancias por sus clases. Las métricas más comunes son:

Entropía:

$$I_e(f) = -\sum_{i=1}^{m} f_i log_2(f_i)$$

Impureza de Gini:

$$I_g(f) = \sum_{i=1}^{m} f_i(1 - f_i)$$

Siendo m el número de clases

Algoritmos de construcción de árboles de decisión como ID3 y C4.5 utilizan la ganancia de información respecto a la entropía de la clase una vez que se conocen los atributos.

Redes Neuronales.

Estructuras constituidas por unidades (neuronas) inspiradas en dos principios fundamentales de las neuronas biológicas:

- Procesamiento de la información de forma distribuida.
- Modificación de las conexiones a medida que se procesa la información.

Las neuronas están conectadas entre sí mediante enlaces con pesos asociados los cuales permiten la ponderación de la señal transmitido por el enlace. Además, cada neurona posee una función de activación que determina si la neurona transmite o no una señal a las neuronas adyacentes. Por lo general, la señal que entra en una neurona n se calcula como $\sum_{i=1}^{n} x_i * w_i$, siendo x_i la señal de la i-ésima neurona conectada a la neurona n y w_i el i-ésimo peso asociado a la conexión por la que se transmite la señal a la neurona n. Llamaremos, yin_n a la señal entrante a la neurona que deberá ser procesada por una función de activación f que determina si la neurona n se activa o no. Existen diferentes funciones de activación, algunas de las más usadas son:

Identidad: $f(yin) = yin$

Escalón: $f(yin) = \begin{cases} 0 \; si \; yin \; < 0 \\ 1 \; si \; yin \; \geq 0 \end{cases}$

Sigmoide: $f(yin) = \frac{1}{1+e^{-yin}}$

ReLU (Rectifier Linear Unit): $f(yin) = \begin{cases} 0 \; si \; yin \; < 0 \\ yin \; si \; yin \; \geq 0 \end{cases}$

Las diferentes estructuras formadas por neuronas permiten el aprendizaje de patrones a través de la modificación de sus pesos, para realizar dicha modificación existen diferentes algoritmos dependiendo de la estructura que forme las neuronas. En relación al número de capas intermedias (capas ocultas) alguna de las reglas para modificar pesos son:

- **Redes monocapa**: algoritmo de aprendizaje de perceptrón y algoritmo de aprendizaje de Adaline (basado en descenso por gradiente).
- **Redes multicapa**: algoritmo *backpropagation* (basado en la retropropagación del error desde la capa de salida a las de entrada pasando por las ocultas).

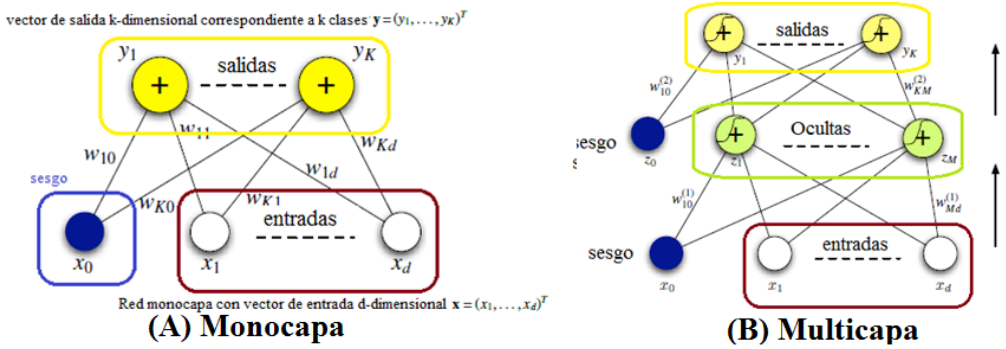


Figura 0-13: Redes Neuronales

Anexo D: Clasificadores sensibles a costes

Un clasificador mapea las *instancias* en *clases.* Ello puede llevarse a cabo con dos tipos de clasificadores según el *tipo de salida* que proporcionan:

1. *discreta*: proporcionan solo una etiqueta que indica la clase predicha para la instancia (p.e. árboles de decisión) y
2. *continua*: estimación de la *probabilidad* (entre 0 y 1) de pertenecer a una clase o un *puntaje*. La asignación a una u otra clase se hace depender de un ***umbral*** de corte para dicho puntaje. Los métodos *bayesianos* y *redes neuronales* pertenecen a este segundo tipo.

Un clasificador *discreto* proporciona solo *un punto* en el diagrama *ROC* y *una sola matriz de confusión*, mientras que otro que funcione *por puntuación* proporcionará un *punto* y una *matriz de confusión* diferente *para cada* ***umbral***.

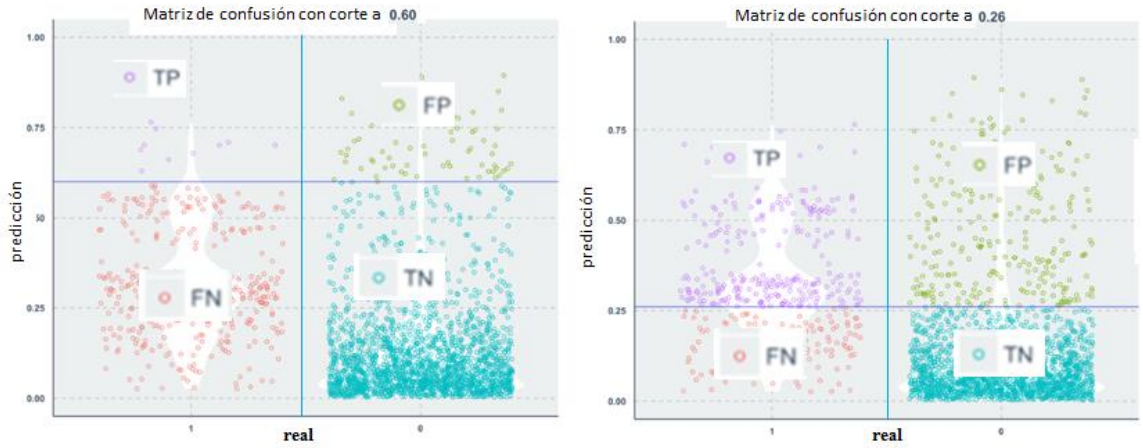


Figura0-1: Matrices de Confusión para Umbrales Distintos

En efecto, si llamamos $umbral = \theta \in R$ tendremos que

$$ROC_x(\theta) = fp(\theta) = \frac{FP(\theta)}{\nu} \qquad \textbf{Ec.(0-2)}$$

$$ROC_y(\theta) = tp(\theta) = \frac{TP(\theta)}{\pi} = 1 - fn(\theta) \qquad \textbf{Ec. (0-3)}$$

Que son las ecuaciones paramétricas de la curva ROC al variar los valores del umbral.
Entre Ec.(0-2) y Ec. (0-3) podemos eliminar θ y obtenemos la ecuación de la *ROC*:

$$ROC_y = F[ROC_x]$$

Si planteamos lo anterior en términos de *contraste de hipótesis en* que *contrastamos que una instancia pertenece a la clase* ***C+***, ***tp*** sería el error tipo I y ***fp*** el error tipo II.
Puede ocurrir que existan, como en medicina o control de fraude, *diferencias entre los costes* de *falsos positivos* y *falsos negativos* y, en este caso, se nos plantea el problema de seleccionar el *umbral de clasificación* más adecuado.

1. Si no existen *costes diferentes de una clasificación errónea* para falsos positivos y falsos negativos, puede enfocarse bien maximizando *F* [Ec.(0-1)] o utilizando el ROC de manera que la *AUC sea lo mayor posible*.

2. Si los costes son diferentes y conocidos, cabe construir una curva de costes a partir de la ROC y seleccionar el umbral de coste mínimo (Figura 0-14).

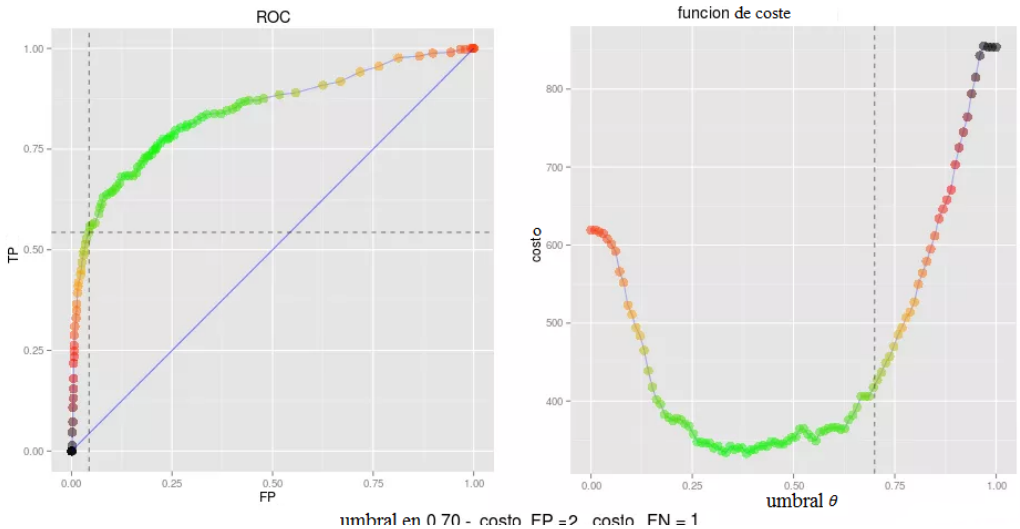


Figura 0-14: ROC y Curva de Costo

Para ello tendremos en cuenta que

$$C(\theta) = C \cdot fp(\theta) + 2C \cdot [1 - tp(\theta)]$$

No obstante, es difícil establecer una relación de costes explícita y, por ello, una alternativa práctica es examinar las *matrices de confusión de umbrales diferentes* y seleccionar entre estos aquel que nos parezca más interesante por su precisión y exhaustividad.

P.e. entre los resultados de un clasificador con dos umbrales distintos[9] y matrices de confusión mostradas en la figura con sus métricas:

		Predicción +	Predicción -	
Real	+	150	30	180
	-	60	20	80
		210	50	260

Métricas	ROC	Usual	
tp	83%	71%	*p*
fp	75%	83%	*r*

Umbral a

		Predicción +	Predicción -	
Real	+	160	20	180
	-	70	10	80
		230	30	260

Métricas	ROC	Usual	
tp	89%	70%	*p*
fp	88%	89%	*r*

Umbral b

Tabla 0-6: Comparación de Umbrales

Es probable que si el usuario da más importancia a la exhaustividad (recuperar el máximo de elementos de la clase *C+*) que a la precisión elegirá el umbral *b* con mayor *r*. Como se ve esto provoca un aumento de los falsos positivos que, por consiguiente, tienen un coste menor que los falsos negativos.

[9] También valdría para comparar dos clasificadores distintos.

Anexo E: Relación de sinonimia

Una dificultad en PLN es la falta de herramientas para idiomas distintos del inglés.
P.e. si queremos obtener sinónimos de una palabra el mecanismo es:
Traducir al inglés→buscar sinónimos en wordnet→ Traducir al español los sinónimos.
Nota: WordNet 3.0es un corpus de sinónimos en inglés.
El siguiente *script* produce sinónimos del array de palabras **`entrada`**:

```
defsinonimia(word):
fromnltk.corpus import wordnet as wn
    b=list()
try:
        a=wn.synsets(word.lower(),lang='spa')
        b=a[0].lemma_names('spa')
fori, word in enumerate(b):
try:
pos=word.index('_')
del b[i]
print(word0)
                word1=word[pos+1:]
word=word0
b.append(word0)
b.append(word1)
except:
pass
else:
                b=a[0].lemma_names('spa')
except:
b.append(word)
        #si no hay sinónimo devuelve la misma palabra
return b

def main():
fromnltk import word_tokenize
entrada=['aeroplano','barco','comida']
for word in entrada:
        t=sinonimia(word)
print('palabra',word.lower(),'sinónimo',t)

if __name__ == '__main__':
main()
```

Lo que produce la siguiente salida:

```
palabra: aeroplano . Sinónimos: ['aeroplano', 'avión']
palabra: barco . Sinónimos: ['barco', 'bote', 'embarcación']
palabra: comida . Sinónimos: ['alimento', 'comida', 'nutriente',
'nutrimento', 'sustento']
```

En cambio, si

```
entrada=['avión','bote','comida']
```

la salida es

```
palabra: avión . Sinónimos: ['avión']
palabra: bote . Sinónimos: ['bote']
palabra: comida . Sinónimos: ['alimento', 'comida', 'nutriente',
'nutrimento', 'sustento']
```

Como puede verse, la relación de sinonimia que se obtiene no es simétrica, depende de las palabras de entrada.

Anexo F: Clasificación con stemming

Durante la fase de desarrollo del modelo de predicción de tuits de odio también se realizaron algunas pruebas de clasificación supervisada utilizando stemming. Una vez obtenido el conjunto de clasificación etiquetado sin haber usado stemming, se procede a la aplicación de esta técnica al corpus y a los atributos de tal forma que a la hora de inicializar las variables de atributos de cada tuit sea más probable encontrar el término en dicho patrón ya que, por ejemplo, las diferentes formas verbales de un término pasan a ser el mismo, los géneros desaparecen y se genera un término común para ambos y lo mismo sucederá con los plurales. Sin embargo, los resultados de clasificación supervisada no fueron tan buenos como se esperaba.
A continuación, se muestra la matriz de confusión obtenida al usar Vecinos Próximos (K-NN) con K=3, distancia de Hamming, variables binarias, 3303 patrones de clasificación y 1803 atributos (al usar stemming la selección de atributos realiza será diferente):

	Matriz de confusión	
No odio	2339	65
Odio	795	104
Real/Predicción	No odio	Odio

La tasa de error mostrada en la matriz es del 26,04% lo que significa que con respecto a 3-NN sin stemming no ha habido mejora alguna. Los valores de precisión (p) y exhaustividad (r) son los siguientes:

$$\boldsymbol{p} = \frac{104}{104+65} = 0.6153 \qquad \boldsymbol{r} = \frac{104}{104+795} = 0.1156$$

La precisión se mantiene igual que en 3-NN sin stemming mientras que la exactitud ha aumentado 0.05 siendo no muy significativa esta variación.
También se han realizado pruebas utilizando BernoullineNB (Naïve Bayes) aplicando stemming, en las mismas condiciones que anteriormente se mencionaron para 3-NN, obteniéndose los siguientes resultados:

	Matriz de confusión	
No odio	2256	148
Odio	292	607
Real/Predicción	No odio	Odio

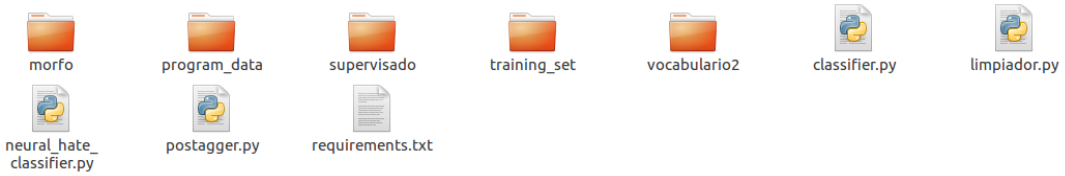
La tasa de error mostrada en la matriz de confusión es del 13,33% un 2% peor que en los experimentos realizados sin stemming. Los valores de precisión (p) y exhaustividad (r) son los siguientes:

$$\boldsymbol{p} = \frac{607}{607+148} = 0.8039 \qquad \boldsymbol{r} = \frac{607}{607+292} = 0.6751$$

Los valores de precisión y exhaustividad también se han reducido. Tras haber realizado estos experimentos llegamos a la conclusión de que el uso de stemming no es útil para enfrentarnos a este problema de clasificación. Este hecho tiene sentido ya que el acortamiento de los términos puede producir conflictos de semántica p.e. retraso y retrasado se con vierten en *retr-* siendo retrasado un término bastante común para insultar

Anexo G: Manual del usuario

El proyecto requiere (§1.2.6) el sistema operativo *Ubuntu 14.4*[10] y *Python 2.7*.
Una vez que se ha separado el clasificador a ser explotado por el usuario del proyecto de investigación, el usuario final tendrá acceso a los siguientes archivos y carpetas:



Una breve descripción de los elementos mostrados en la imagen:

- Carpeta "`morfo`": contiene las reglas del *POS-Tagger* necesarias para determinar la categoría de las palabras de los tuits a clasificar.
- Carpeta "`program_data`": contiene tf-idfs de los términos previamente calculados y un diccionario que permite la transformación de términos en identificadores.
- Carpeta "`supervisado`": contiene clases necesarias para la extracción de los datos del conjunto de entrenamiento.
- Carpeta "`training_set`": contiene el conjunto de entrenamiento de 3.303 patrones, 2.532 atributos y variables tf-idf.
- Carpeta "`vocabulario2`": contiene las palabras utilizadas para la construcción del filtro de tuits.
- "`limpiador.py`": módulo encargar del preprocesamiento del tuit.
- "`postagger.py`": módulo en el que está contenido el etiquetador de palabras (pos-tagger)
- "`neural_hate_classifier.py`": módulo que contiene el perceptrón multicapa de 3 capas ocultas con el que se obtuvo el mejor resultado de validación.
- "`classifier.py`": módulo que contiene el clasificador a ser explotado por el usuario final.
- "`requeriments.txt`": contiene el nombre de las bibliotecas necesarias que el usuario final debe instalar.

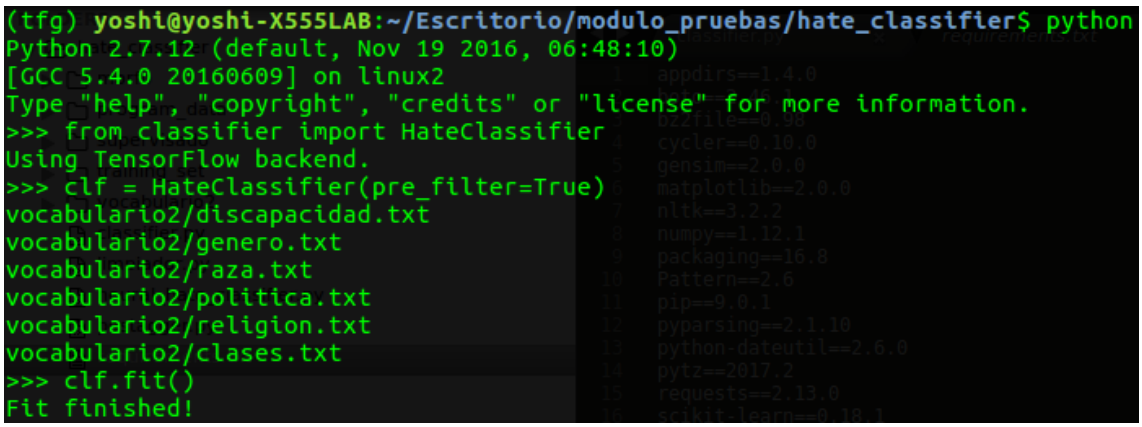
Para instalar las bibliotecas necesarias se deberá hacer uso del comando en sistemas Linux:
`"pip install -r requeriments.txt"`
Tras haber instalado las bibliotecas necesarias contenidas en `requeriments.txt`, el usuario podrá importar la clase *HateClassifier* situada en el módulo "`classifier.py`". De esta forma se pueden crear instancias del clasificador de tuits siendo posible indicar en el constructor de la clase si se quiere utilizar o no el filtro.
Una vez creada la clase hay dos métodos de interés para el usuario:

- ***Fit***: permite realizar el entrenamiento de la red neuronal.
- ***Predict***: permite clasificar una cadena de texto pasada por parámetro una vez que se ha realizado previamente el entrenamiento del clasificador. Este método puede devolver tres valores:

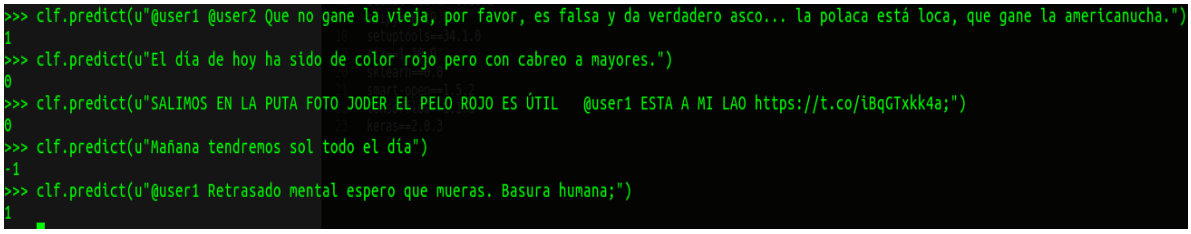
[10] Si no se usa el POS- Tagger, eliminando stopwords, p.e., es posible correr los programas sobre *Windows*.

- 0 si el tuit no contiene odio,
- 1 si el tuit contiene odio a una persona o colectivo y
- -1 si no ha pasado el filtrado previo.

Una pequeña demo de utilización del clasificador podría ser la siguiente:



En primer lugar, se importa la clase *HateClassifier* y se lleva a cabo la instanciación) de la misma (`clf= HateClassifier(pre_filter=True)`, especificando que se utiliza filtrado previo a la clasificación. A continuación, se mostrarán todos los archivos que contiene el vocabulario del filtro. Tras la instanciación se realiza el entrenamiento del clasificador.



Una vez entrenado el clasificador, este se puede utilizar para clasificar todos los tuits que se deseen, únicamente haciendo uso del método `predict` e introduciendo como parámetro el texto del tuit.