Escuela Politécnica Superior

Distinción de bots y humanos en Twitter con Inteligencia Artificial

Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Autor:
María Esther Rico Martínez

Tutor/es:
José Vicente Berná Martínez

Universitat d'Alacant
Universidad de Alicante

Diciembre 2021

Resumen

En la actualidad, las personas cada vez hacemos más uso de las redes sociales, llegando muchas de ellas a convertirse en medios informativos y lugares donde las personas pueden expresar y compartir sus opiniones, propagándose toda esta información con gran velocidad. Así, esta facilidad de propagación que tienen las redes sociales ha despertado un problema de especial relevancia que es la rapidez con la que proliferan y se difunden contenidos falsos o malintencionados en estas redes. Estos contenidos en muchas ocasiones tienen detrás de si un perfil falso o controlado por un bot con intenciones maliciosas. Sin embargo, hoy por hoy, muchas de estas cuentas sospechosas se camuflan imitando a una persona real, hecho que dificulta enormemente su identificación. Por ello, antes de creer y confiar en cualquier contenido que se muestre en este tipo de redes, es necesario contrastar dicha información y valorar la credibilidad que transmite su autor.

Así, con la elaboración de este trabajo, se pretende aportar una experimentación formal y extensa, focalizada en lograr la distinción entre personas reales y bots en la red social Twitter, empleando para ello técnicas de Inteligencia Artificial basándose en sus características de perfil. Estos datos presentes en los perfiles de cada cuenta se obtienen mediante la API de Twitter y el lenguaje Python junto con el uso de varias librerías. De esta forma, se recopila un conjunto amplio de usuarios sobre los cuales se conoce previamente si son bots o humanos y se extraen todas las características de dichos perfiles

Una vez obtenidas dichas características, se procede a aplicar diferentes técnicas de Machine Learning para determinar qué datos de los extraídos cobran una mayor peso a la hora de identificar si se trata de una cuenta controlada por una persona real o por un bot.

Seguidamente, se procede a hacer uso de soluciones basadas en Deep Learning, definiendo para ello modelos de redes neuronales con distintas configuraciones. En esta experimentación se realiza un estudio exhaustivo probando diversas estructuras de modelos con el fin de buscar aquellas que, una vez se entrenen utilizando como entrada las características de los usuarios, sean capaces de predecir de forma óptima si detrás de una cuenta hay un bot o un humano. Sin embargo, esta tarea no será sencilla, pues la heterogeneidad de los datos extraídos para cada uno de los perfiles hará que el proceso de aprendizaje de los modelos se vuelva complejo.

Motivación, justificación y objetivo general

El motivo principal que me ha impulsado a llevar a cabo el desarrollo de este proyecto resulta ser mi interés por la minería y el procesamiento de datos, así como la inquietud por aprender sobre nuevas tecnologías en auge como la Inteligencia Artificial. Considero que disponer de grandes volúmenes de información no sirve de nada si no conseguimos sacarles partido.

Hoy en día, las redes sociales se han convertido en el método informativo que muchas personas consultan de manera diaria debido a su inmediatez y comodidad. Sin embargo, la rápida propagación de contenidos en estas redes dificulta la tarea de identificar contenidos falseados o con intención de desinformar o engañar. Por tanto, los usuarios que se informan con estas redes deberían someter a los autores de las publicaciones a un análisis de credibilidad previo, con el fin de intentar descubrir si pueden fiarse o no del contenido leído. Pero esta tarea no es nada sencilla, pues muchas veces las cuentas falsas o controladas por bots intentan simular comportamientos humanos con el fin de ganar credibilidad y pasar desapercibidas.

Por ello, me gustaría focalizar mi proyecto en la búsqueda de soluciones que contribuyan en la distinción entre cuentas controladas por bots y humanos, ya que considero que es una tarea muy compleja a la que deberíamos de hacer frente los usuarios de estas plataformas de forma diaria, a fin de no caer en la desinformación o en el engaño. Creo que el desarrollo de este proyecto me va a permitir ir un paso más allá y plasmar gran parte de mis conocimientos adquiridos tanto a lo largo del grado como del máster, pues considero que toda persona tiene derecho a informarse sin tener que someter a juicio personal cada publicación que lea.

Para frenar la propagación de este tipo de contenidos fraudulentos, es necesario identificar a sus autores. Por tanto, aportar con mi trabajo una ayuda que contribuya a combatirlo, me resulta muy gratificante como usuaria de estas plataformas.

Agradecimientos

En primer lugar, me gustaría agradecerle a mi tutor de TFM, José Vicente Berná, toda la disposición, paciencia y dedicación que ha depositado en este proyecto para que pudiera llevarse a cabo. Agradezco enormemente todas las ayudas y consejos que me has dado, pues has logrado guiar mi trabajo enormemente y me has hecho muy ameno todo el desarrollo del mismo.

Además, querría agradecer en particular a dos personas muy importantes en mi vida. En primer lugar, a mi padre por todo el apoyo que me ha brindado continuamente a lo largo de las distintas fases de mi proyecto, animándome en todo momento y ayudándome en los momentos más complejos y difíciles.

En segundo lugar, a mi mejor amiga, Paula, que más que una amiga es una hermana para mí. Gracias por estar siempre en mi vida, en las buenas, en las malas y en las peores. Gracias por todo el apoyo constante que me has brindado, amiga. Gracias por ser como eres. Gracias por tener siempre tanta luz. Gracias por todas las risas y sonrisas que me regalas siempre, incluso en momentos difíciles. Y sobre todo, gracias por ser mi mejor amiga.

Citas

Mi comprensión solo puede ser una fracción infinitesimal de todo lo que quiero entender.

Ada Lovelace

Pensé más en que: ¡por Dios, el software funcionó!; que en que habíamos llegado a la Luna.

Margaret Hamilton

Conseguir un objetivo te proporciona una satisfacción inmediata; el proceso para conseguirlo es un placer duradero.

Evelyn Berezin

Los seres humanos son alérgicos a los cambios. Les encanta decir: "Siempre lo hemos hecho así". Trato de luchar contra eso. Es por eso por lo que tengo un reloj en la pared que funciona en sentido contrario.

Grace Murray Hopper

Me gusta aprender. Eso es un arte y una ciencia.

Katherine Johnson

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de códigos

1. Introducción

Hoy en día cada vez es más evidente el desorden informacional ante el que estamos expuestos los usuarios en la red, hecho que se ha visto potenciado aún más gracias al avance de tecnologías como internet y a la mayor accesibilidad de las personas a dispositivos como los smartphones, pues facilitan la rápida propagación de este tipo de contenidos erróneos o creados con fines específicos como sería engañar o influir sobre la opinión pública [1].

Uno de los focos más importantes de difusión informativa son las redes sociales, cuya veracidad y credibilidad muchas veces se pone en entredicho debido a las labores desinformativas que desarrollan una parte de usuarios dentro de las mismas y a su rápida propagación. Así pues, cuando se comienza a viralizar en poco tiempo un tema específico y existe por tanto una sobreabundancia de información sobre ello, ya sea falsa o verdadera, surge el término infodemia [2]. Un claro ejemplo de ello es la gran cantidad de información que comenzó a circular en la red sobre la pandemia actual de COVID-19, especialmente en redes sociales como Twitter, donde la desinformación ha cobrado un gran protagonismo, generando consigo malestar, incertidumbre y miedo entre los usuarios. Este hecho ha agravado consigo la situación ya que, ante estas ingentes cantidades de información, resulta complejo para los usuarios localizar fuentes fiables de información donde poder contrastar todos estos datos [3]. Por ello, aportar soluciones que colaboren en la determinación de la credibilidad de los usuarios activos que intervienen en estas redes sociales, resulta crucial a la hora de localizar aquellas cuentas que, lejos de participar de forma sana en la red social, con una finalidad divulgativa y/o meramente informativa, se dedican a difundir y propagar información falsa en la plataforma con dudosas intenciones.

En el año 2019 realicé un trabajo de fin de grado titulado "*Análisis de la credibilidad de cuentas de Twitter"* [4], en el cual ya estuve trabajando sobre estos aspectos anteriormente comentados. La idea de este proyecto se basaba en el desarrollo de una herramienta que permitiera valorar la credibilidad que presentaba un usuario de la red social Twitter ideando para ello un algoritmo que, en base a unos factores determinantes y no determinantes que establecí, generaba una puntuación final que reflejaba la credibilidad de cada una de las cuentas analizadas. Asimismo, este algoritmo diseñado se demostró que cobraba una mayor eficacia si se observaba en el tiempo, es decir, si se estudiaba la tendencia de la credibilidad de los usuarios transcurrido un periodo. Estos resultados obtenidos permitían distinguir, por un lado, lo que se categorizó como cuentas buenas, donde participaban personas o entidades reales y, por otro lado, cuentas malas, que resultaban sospechosas o controladas por algún tipo de software.

Sin embargo, las redes sociales son un mundo muy cambiante y hoy en día existen muchos más elementos interesantes que pueden ayudar en la clasificación de la credibilidad de las cuentas de estas plataformas que, o bien no existían, o bien no cobraban la importancia merecida por aquel entonces. Así pues, cada vez es más frecuente localizar diversos tipos de trampas o formas de manipulación en los contenidos presentes en redes sociales con diversos fines como desinformar o influenciar masivamente. Desde hace años, se localizan cuentas controladas por bots que, gracias al uso de la automatización, consiguen simular el comportamiento de una persona real en estas plataformas con el fin de pasar desapercibidos, hecho que dificulta enormemente su identificación [5].

Así, en este presente trabajo se pretende continuar con la línea de investigación que comencé en el proyecto de mi trabajo de fin de grado, con el fin de seguir estudiando factores que influyan y/o determinen la credibilidad de los usuarios en la red social Twitter y elaborar así una herramienta mucho más precisa, fiable y refinada que la desarrollada con anterioridad.

La idea es por tanto colaborar en la protección del contenido real y de calidad presente en las redes sociales, acabando para ello con aquellos focos divulgativos de noticias falsas que, lejos de ser objetivos, pretenden influir en la opinión popular y propagar información tendenciosa o falseada con diversos propósitos. Los usuarios de estas plataformas deben de poder hacer uso de las mismas disponiendo de contenido veraz y sin tener la necesidad de cerciorarse sobre la credibilidad de los autores de las publicaciones que vayan descubriendo mientras hacen uso de estas redes sociales.

2. Planificación

La planificación temporal de cada uno de los bloques principales de este trabajo se detalla en la *Tabla 1* localizada en este apartado. Así pues, se distinguen tres bloques principales sobre los cuales girará el desarrollo de este proyecto:

- **En primer lugar,** se procederá a detallar la motivación que ha conducido al desarrollo de este trabajo, los asuntos más relevantes sobre los que se pretende indagar, así como la trascendencia y utilidad que puede derivar de todo ello. Tras ello, se continuará abordando la introducción del proyecto con el fin de contextualizarlo y enmarcarlo en el entorno social actual.

  Seguidamente, se comenzará el apartado de antecedentes, donde se hará un breve resumen del trabajo previamente realizado como Trabajo de Fin de Grado y sobre el cual se sustentará el desarrollo del actual. Una vez detallada la investigación de la cual se parte, se procederá a iniciar el apartado del estado del arte, en el cual se investigará sobre todos los aspectos que influyen en la problemática actual, las tecnologías que se utilizan, así como investigaciones y proyectos previos de otros autores, a los que se pueda hacer alusión para empezar.
- **En segundo lugar**, tras finalizar la investigación previa sobre la problemática y las soluciones existentes en la actualidad, se establecerán los objetivos a abordar en el presente proyecto. De esta forma, una vez definidos se procederá a establecer una planificación realista y coherente que permita su correcto desarrollo.

  Tras ello, se realizará una propuesta de solución que cubra los objetivos descritos y que, seguidamente, deberá de validarse. Este proceso de validación requerirá de una recogida de datos que tendrá una duración de 1 mes.
- Finalmente, **en tercer lugar**, se continuará efectuando la implementación del código y su posterior validación, con el fin de asegurar que funciona de manera correcta. Así, una vez se verifique su correcto funcionamiento, se realizarán determinadas pruebas con el fin de asegurar este funcionamiento. De esta forma, para la realización de estas pruebas será necesario realizar una recogida de datos durante un periodo de tiempo de 1 mes.

  Tras ello, se procederá a extraer las conclusiones de los resultados obtenidos tras las pruebas realizadas, así como posibles mejoras o trabajo pendiente que pudiera tenerse en cuenta para el futuro. Por último, se rellenarán los apartados de agradecimientos y

citas, así como se revisará que todas las referencias presentes en el documento final se encuentran debidamente enlazadas.

Tabla 1. Planificación temporal TFM

Contenidos	**Tiempo específico**	**Tiempo total**	**Fecha límite fin**
Motivación, justificación y objetivo general Introducción Antecedentes Estado del arte		1 mes y medio	30 junio
Objetivos Planificación Propuesta y validación		2 meses	1 septiembre
• Recogida de datos para la validación	1 mes		
Implementación y validación		2 meses	15 noviembre
• Recogida de datos para las pruebas	1 mes		
Pruebas de la propuesta Conclusiones y trabajo futuro Referencias, bibliografía y apéndices Agradecimientos, citas, índices			

3. Antecedentes

Este proyecto se plantea a raíz del trabajo de fin de grado que realicé previamente en 2019 titulado "*Análisis de la credibilidad de cuentas en Twitter"* [4].

3.1. Estado del arte previo

La línea de investigación de este proyecto comenzó analizando la gran desconfianza existente sobre la información que circula en las redes sociales. Este hecho se debe a la existencia de contenido manipulado o falseado en las mismas que, sumado a la posibilidad de viralizar cualquier información en estas plataformas, supone un problema serio, especialmente cuando se pretende desinformar o influir sobre la opinión pública mediante trampas o engaños. Además, en general las noticias falsas resultan más llamativas y atractivas para los usuarios, hecho que incentivaba enormemente su difusión, tal y como se pudo confirmar en un estudio realizado por el Instituto de Tecnología de Massachusetts (MIT) [6].

Por ello, resulta realmente importante continuar con la búsqueda de soluciones o estrategias que ayuden a frenar esta desinformación, algo en lo que trabajan constantemente las propias redes sociales como WhatsApp, limitando la cantidad de veces que un mensaje puede ser reenviado [7], Twitter, condicionando la difusión de cada publicación según el comportamiento del entorno social de su autor [8] o Facebook, mediante el uso de servicios verificadores de información [9], entre otras. Sin embargo, es necesario que los usuarios de estas plataformas colaboren también en esta labor, analizando de forma crítica la información circulante y reportando aquellas noticias o contenidos que, una vez contrastados en fuentes fiables, se confirme que han sido manipulados y no se corresponden con la realidad. Así, esta gran capacidad de propagación de contenidos que existe en las redes sociales ha propiciado la aparición de cuentas falsas, algunas de ellas incluso controladas por un software programado que simula actitudes y comportamientos humanos con el fin de pasar desapercibidas.

Así pues, se decidió investigar en particular sobre la red social Twitter, una de las redes sociales con mayor cantidad de usuarios activos en el mundo y cuyo objetivo principal se basa en la creación y la difusión de contenidos en publicaciones conocidas con el nombre de tuits. De esta forma, la posibilidad que ofrece esta plataforma para compartir contenidos en tiempo real junto con su rápida propagación ha hecho que sea una de las redes sociales más atractivas tanto para la prensa como para diversas empresas que han descubierto en ella una nueva forma de publicitarse y/o captar la opinión pública.

Sin embargo, no toda la información publicada en esta red social resulta fiable, pues en ocasiones puede ser ficticia o errónea, hecho que puede derivar en un gran impacto negativo sobre los usuarios cuando se difunde información fraudulenta o de dudosas intenciones, llegando a generar pánico o reacciones hostiles sobre los lectores.

De esta forma, con el paso del tiempo han surgido numerosas iniciativas que buscan ayudar a detectar y a frenar esta propagación descontrolada de contenidos falsos o maliciosos, las cuales se centran especialmente en los siguientes aspectos, empleando técnicas muy diversas:

- Realizar un análisis exhaustivo de los metadatos del usuario que haya compartido un determinado contenido.
- Determinar el nivel de impacto de la publicación en la red social en base a su repercusión, analizando para ello metadatos como el número de veces que se ha compartido o la cantidad de favoritos recibidos, entre otros.
- Estudiar en profundidad la lingüística presente en el tuit con el fin de determinar la presencia o no de elementos que puedan expresar sentimientos, opinión, engaño o intención de desinformar.
- Observar el comportamiento del usuario autor de la publicación, su influencia y sus interacciones sociales en la plataforma.
- Analizar aspectos del usuario con el fin de determinar si se trata de una cuenta controlada por algún tipo de software (bot) o de lo contrario existe una persona real detrás de la misma.

En definitiva, el aspecto central sobre el que recae una mayor atención resulta ser el usuario en sí, es decir, en concreto el grado de credibilidad que presenta dicha cuenta en la plataforma y que para ello pueden tenerse en cuenta aspectos tan diversos como su información personal, su comportamiento o la influencia que ejerce en la red social, entre otros. Todos estos aspectos es posible estudiarlos mediante el análisis de los metadatos vinculados tanto al usuario como a sus publicaciones.

Para poder acceder a estos datos que resultan de interés, Twitter ofrece acceso a la información de aquellas cuentas que dispongan de un perfil público a través de las interfaces de programación de aplicaciones (API) que facilita. Con ellas, pretenden así posibilitar y favorecer el desarrollo de nuevas aplicaciones que puedan integrarse con esta red social. Así pues, para tener acceso a dicha API es necesario crear una cuenta de desarrollador que deberá ser autorizada, ya que Twitter pretende controlar de manera individualizada el uso que se hace de su contenedor de datos.

Con respecto al desarrollo del proyecto, tras leer la documentación oficial de la API de Twitter, se decidió escoger el lenguaje de programación Python pues resultaba ser uno de los lenguajes con mayor cantidad de librerías dedicadas a esta plataforma. Así pues, se trata de un lenguaje muy popular y utilizado en la actualidad, el cual presenta numerosas ventajas entre las que se pueden destacar la sencillez de su sintaxis, su compatibilidad con diversas plataformas, su libre distribución, la gran cantidad de librerías de las que dispone, así como el uso de "machine learning" en algunos de sus paquetes para el procesamiento de grandes volúmenes de información con una gran precisión y exactitud, entre otras.

De esta forma, entre las librerías existentes para la red social Twitter, se destacaron tres de ellas en especial, las cuales facilitaban la extracción de la información pública de la plataforma como contenedores de información: Tweepy, Twython y TwitterAPI. Así pues, tras analizar en profundidad las ventajas, posibilidades y funcionalidades que ofrecían cada una de las mismas, se decidió optar por la librería Tweepy. Esta librería presentaba múltiples ventajas: era la más popular entre la comunidad de desarrolladores, resultaba muy intuitiva y sencilla de utilizar y, además, disponía de un gran abanico de métodos para utilizar sobre la API de Twitter, una gran cantidad de tutoriales en la red sobre cómo utilizarla, así como una documentación abundante y muy detallada.

3.2. Algoritmo propuesto y validación

Para determinar la credibilidad de una cuenta de Twitter se propuso una fórmula donde se tenían en cuanta diversos indicadores que se alimentaban de características públicas de los perfiles de los usuarios. Entre los indicadores que se establecieron, no todos poseían la misma relevancia, distinguiendo así entre factores determinantes F_d, aquellos que resultaban más decisivos, y factores no determinantes F_{nd}, aquellos que eran menos concluyentes. De esta forma, teniendo en cuenta estos indicadores, el algoritmo generaba una puntuación final vinculada a la cuenta $P(CC_t)$, donde cuanto mayor fuera la puntuación obtenida para ese usuario, mayor credibilidad se asociaba a dicho perfil.

Así pues, la fórmula para obtener la puntuación final sobre la credibilidad de las cuentas que se planteó fue la siguiente:

$$P(CC_t) = |1 + \sum_{i=1}^{n} \big(P(F_{nd})\big)| \cdot \prod_{j=1}^{m} (P(F_d))$$

Donde:

$\{P(F_{nd_1}), P(F_{nd_2}), \dots P(F_{nd_n})\}$ – Puntuaciones asociadas a factores no determinantes.

$\{P(F_{d_1}), P(F_{d_2}), \dots P(F_{d_m})\}$ – Puntuaciones asociadas a factores determinantes.

De esta forma, tal y como se observa en la fórmula anterior, la puntuación final se conforma, entre otros aspectos, por el sumatorio de las puntuaciones de los factores no determinantes, pues no resultan tan relevantes, y el producto de los factores determinantes, ya que resultan más decisivos y por tanto se le otorga un mayor peso a la hora influir en la puntuación final. Para evitar que el bloque del sumatorio pudiera tener un valor 0 y que anulase el producto posterior, se decidió sumar siempre una unidad y sacar posteriormente el valor absoluto de esa parte.
Así, si se desglosan las puntuaciones de los indicadores que conforman ambos grupos de factores, determinantes y no determinantes, la fórmula anterior quedaría de la siguiente manera:

$$P(CC_t) = \Big|1 + \big(P(R_d) + P(T_d) + P(S_u) + P(R_{ss}) + P(A_c) + P\big(D_p\big) + P\big(L_p\big) + P(N_l) + P(T_a) + P\big(P_p\big) + P(P_b)\big)\Big| \cdot (P(V_c) \times P(I_p))$$

Donde:

<u>Factores no determinantes</u>

$P(R_d)$ – Puntuación vinculada a la cantidad de retuits diarios realizados por el usuario.

$P(T_d)$ – Puntuación asociada a la cantidad de tuits diarios efectuados por el usuario.

$P(S_u)$ – Puntuación conforme a la cantidad de seguidores que tiene el usuario.

$P(R_{ss})$ – Puntuación de acuerdo con el ratio seguidores/seguidos que posee el usuario.

$P(A_c)$ – Puntuación vinculada a la antigüedad de la cuenta del usuario.

$P\big(D_p\big)$ – Puntuación conforme a la existencia o falta de una descripción en el perfil del usuario.

$P\big(L_p\big)$ – Puntuación con arreglo a la presencia o ausencia de la localización geográfica en el perfil del usuario.

$P(N_l)$ – Puntuación asociada a la cantidad total de "likes" realizados por el usuario.

$P(T_a)$ – Puntuación vinculada a la media de publicaciones diarias según la antigüedad de la cuenta del usuario medida en días.

$P\big(P_p\big)$ – Puntuación conforme a la existencia o ausencia de personalización en el diseño del perfil del usuario.

$P(P_b)$ – Puntuación de acuerdo con el resultado arrojado por Botometer sobre la cuenta del usuario.

<u>Factores determinantes</u>

$P(V_c)$ – Puntuación conforme a si la cuenta del usuario está verificada o no.

$P(I_p)$ – Puntuación asociada a la presencia o falta de una imagen de perfil escogida por el usuario.

Cada una de estas puntuaciones se calculaban de forma distinta entre sí, obteniendo así un valor resultante asociado al indicador correspondiente en cada caso. Una vez obtenidas estas puntuaciones para cada uno de los indicadores, se sustituían los valores en el algoritmo ideado y se efectuaban las operaciones con el fin de obtener así la puntuación total sobre la credibilidad de cada cuenta. Sin embargo, este valor cobraba especial relevancia si se estudia en el tiempo, es decir, observando la tendencia de la credibilidad con el transcurso de los días. De esta forma, se plantearon los siguientes escenarios según fuera la tendencia obtenida:

- *Tendencia positiva o muy positiva*, indicio que señalaba un hecho poco frecuente que podía derivar del empleo de estrategias de marketing o comerciales, campañas publicitarias o tácticas de manipulación, entre otros.
- *Tendencia constante o plana (con poco cambio)*, predisposición asociada a cuentas que pertenecieran a usuarios reales con poco cambio.
- *Tendencia negativa o muy negativa*, aspecto que sugería que la cuenta se podía estar abandonado o dejando de usar.

Esta clasificación de la tendencia fue totalmente especulativa, pues únicamente se automatizaron los cálculos correspondientes a las puntuaciones de cada uno de los factores y la puntuación final de la credibilidad.

De esta forma para cada uno de los factores, tanto determinantes como no determinantes, se consultaron diversos estudios e investigaciones con el fin de identificar qué situaciones para cada factor suponían un aumento en la credibilidad de la cuenta y qué aspectos por el contrario la penalizaban. Con ello, se establecieron distintos rangos para cada indicador y, dependiendo del resultado obtenido, se asignaba más o menos peso a su puntuación según su influencia sobre la credibilidad del usuario.

Así pues, de manera resumida, los rangos de puntuación establecidos para cada uno de los factores resultaban ser los siguientes:

Tabla 2. Rangos de puntuación para los factores no determinantes

Factores no determinantes		
Descripción del factor	**Condición**	**Puntuación**
Cantidad de retuits diarios realizados por el usuario	0 < cantidad_ retuits ≤ 10	$P(R_d) = 100$
	cantidad_retuits > 10	$P(R_d) = 50$
	cantidad_retuits = 0	$P(R_d) = 0$
Cantidad de tuits diarios realizados por el usuario	0 < cantidad_ tuits ≤ 20	$P(T_d) = 100$
	cantidad_ tuits > 20	$P(T_d) = 50$
	cantidad_ tuits = 0	$P(T_d) = 0$
	cantidad_seguidores ≥ 707	$P(S_u) = 100$

Cantidad de seguidores que tiene el usuario	0 < cantidad_seguidores < 707	$P(S_u) = 50$
	cantidad_seguidores = 0	$P(S_u) = 0$
Ratio seguidores/seguidos que posee el usuario	$ratio\left(\frac{seguidores}{seguidos}\right) > 2$	$P(R_{ss})$ = 100
	$0.75 \leq ratio\left(\frac{seguidores}{seguidos}\right) \leq 2$	$P(R_{ss})$ = 50
	$ratio\left(\frac{seguidores}{seguidos}\right) < 0.75$	$P(R_{ss})$ = 0
Antigüedad de la cuenta del usuario	antigüedad_cuenta ≥ 1 año	$P(A_c) = 100$
	antigüedad_cuenta < 1 año	$P(A_c) = 0$
Descripción en el perfil del usuario	tamaño_cadena(descripción) > 0	$P(D_p) = 100$
	tamaño_cadena(descripción) = 0	$P(D_p) = 0$
Localización geográfica del usuario	tamaño_cadena(localización_geográfica) > 0	$P(L_p) = 100$
	tamaño_cadena(localización_geográfica) = 0	$P(L_p) = 0$
Diseño del perfil del usuario	personalización_por_defecto = FALSE	$P(P_p) = 100$
	personalización_por_defecto = TRUE	$P(P_p) = 0$
Cantidad total de "likes" dados por el usuario	numero_likes ≥ 200	$P(N_l) = 100$
	0 < numero_likes < 200	$P(N_l) = 50$
	numero_likes = 0	$P(N_l) = 0$
Media de publicaciones diarias en base a la antigüedad del usuario (días)	$15 \leq$ media_publicaciones_diarias ≤ 30	$P(T_a) = 500$
	$5 \leq$ media_publicaciones_diarias < 15	$P(T_a) = 100$
	$30 <$ media_publicaciones_diarias ≤ 40	$P(T_a) = 100$
	media_publicaciones_diarias < 5	$P(T_a) = 0$
	media_publicaciones_diarias > 40	$P(T_a) = 0$
Resultado para el usuario en Botometer	nota_botometer ≤ 1	$P(P_b) = 500$
	nota_botometer ≤ 5	$P(P_b) = 0$

Tabla 3. Rangos de puntuación para los factores determinantes

Factores determinantes		
Descripción del factor	**Condición**	**Puntuación**
Verificación de una cuenta de usuario	Cuenta verificada	$P(V_c) = 1000$
	Cuenta no verificada	$P(V_c) = 1$
Personalización de la foto de perfil	Foto de perfil personalizada	$P(I_p) = 1$
	Foto de perfil por defecto	$P(I_p) = \frac{1}{1000}$

Una vez definido el algoritmo en detalle, se llevó a cabo su validación, utilizando para ello grupos de datos controlados con los que poder verificar que los valores obtenidos tras aplicar la fórmula coincidiesen con los esperados. Así pues, para ello, se seleccionó un grupo de cuentas representativas de diferentes tipos de perfiles de usuarios, de las cuales ya se conocía previamente su clasificación. Con ello, sería posible controlar que el algoritmo ideado realizaba adecuadamente la clasificación del listado de cuentas, pudiendo así validar su correcto funcionamiento.

De esta manera, se planteó la siguiente clasificación para los distintos tipos de cuentas:

- *Cuentas buenas*, donde se ubicaba a perfiles de usuarios reales, tanto verificados como no verificados.
- *Cuentas malas*, donde se localizaban a perfiles de usuarios falsos o controlados por software.

Así pues, seguidamente, se escogieron cuentas de la red social Twitter que coincidieran con la clasificación definida con anterioridad, seleccionando para ello cuentas que presentasen propiedades que justificasen su pertenencia a un tipo u otro. En este caso se escogieron una muestra total de 11 cuentas, de las cuales 6 pertenecían al tipo "cuentas buenas" y 5 al de "cuentas malas".

Tras seleccionar las cuentas a analizar, fue necesario implementar un fragmento de código muy escueto que permitiera efectuar la extracción diaria de los datos de cada una de estas cuentas, accediendo para ello a la API de Twitter. Esta información de cada uno de los perfiles fue recopilada de forma diaria durante un periodo total de 30 días, lo que generó un total de 330 registros totales. Así, una vez se obtuvo toda la información de interés, se procedió a calcular la puntuación final $P(CC_t)$ diario para cada una de las cuentas seleccionadas. En consecuencia, al calcular la puntuación diaria de cada cuenta durante un periodo tan importante de días, era posible obtener una tendencia especialmente fiable sobre la credibilidad.

Seguidamente, se aplicó de forma manual el algoritmo a los registros recopilados con el fin de observar la evolución de la tendencia de cada cuenta en el tiempo. De esta manera, los resultados obtenidos fueron los siguientes:

- En el caso de las *cuentas buenas*, por un lado, las verificadas presentaban puntuaciones muy altas y una actividad constante con picos puntuales de subidas, hecho que podía derivar de algún tipo de campaña publicitaria o de marketing al tratarse de empresas o personas de especial relevancia. Por otro lado, las no verificadas mostraban también variaciones en su puntuación, siendo estos cambios de actividad en los usuarios reales algo normal y frecuente, pudiendo influir además aspectos muy diversos como la época del año, los periodos vacacionales, así como situaciones personales o laborales, entre otros. De esta forma, la credibilidad a lo largo del tiempo para estos usuarios nunca sería una línea horizontal de puntos idílica, pues la aleatoriedad humana siempre estaría presente.
- En el caso de las *cuentas malas*, se observó que solían presentar puntuaciones bastante pobres y, además, la línea dibujada de la tendencia de su credibilidad formaba una línea horizontal casi perfecta. Así, tras revisar en detenimiento dicho resultado, se detectaron

conductas idénticas y estáticas durante todo el mes, hecho que distaba mucho de corresponderse con comportamientos de usuarios reales. Asimismo, se detectaron cuentas que presentaban una puntuación extremadamente baja, hecho que podía indicar que se estaban dejando de usar o se habían abandonado.

3.3. Pruebas realizadas y evaluación

Tras realizar la implementación del algoritmo diseñado y efectuar su validación, se procedió a probar el mismo, con el fin de poner a prueba su validez y observar los resultados obtenidos con el fin de determinar el grado de relevancia que arrojaban estas puntuaciones finales sobre cada una de las cuentas sujetas a estudio.

En este caso, puesto que coincidió en el tiempo del desarrollo del proyecto el escenario de las elecciones electorales de noviembre de 2019, se optó por estudiar la calidad del usuario medio que existía entre los seguidores de varios partidos políticos en sus respectivas cuentas de Twitter. De esta forma, se seleccionó un conjunto de N cuentas que seguían en la red social a los partidos políticos seleccionados con el fin de calcular el $P(CC_t)$ medio de estos seguidores, así como estudiar su evolución en un periodo de tiempo con el fin de evaluar como de buenas o malas eran estas cuentas y determinar su grado de credibilidad. Así, para obtener la media de las puntuaciones de credibilidad asociadas a cada cuenta de la muestra tomada, se optó por efectuar la media aritmética con el fin de facilitar el análisis posterior.

De esta manera, se seleccionaron de manera manual las 100 primeras cuentas que seguían a cada uno de los partidos políticos PSOE, PP, Podemos y Vox en la red social. De esta forma, se obtuvo un total de 400 cuentas a analizar, sobre las cuales se recopiló 1 registro diario durante una semana, haciendo un total de 2800 registros totales a estudiar.

Así, los resultados obtenidos sobre la puntuación de credibilidad media que presentan los seguidores de cada partido político resultaron ser los siguientes:

- En primer lugar, en el caso de los resultados vinculados al $P(CC_t)$ de los seguidores de la cuenta de Twitter del PSOE, se observó una tendencia lineal que comenzaba a decrecer a partir del mismo día de las elecciones. Este hecho podía indicar un descenso en la actividad de sus seguidores tras haber tenido ya lugar las elecciones y/o la presencia de bots o cuentas falsas entre los usuarios. Por tanto, estos usuarios fueron catalogados como “cuentas malas”.

- En segundo lugar, en el caso de los resultados vinculados al $P(CC_t)$ de los seguidores de la cuenta de Twitter del PP, se contempló una tendencia lineal con un leve decrecimiento y valores muy dispares entre los distintos días de medición. Este descenso era muy leve en la puntuación final de sus seguidores y, por tanto, en valores generales su tendencia se mantuvo estable. Por tanto, estos usuarios fueron catalogados como "cuentas buenas".
- En tercer lugar, en el caso de los resultados vinculados al $P(CC_t)$ de los seguidores de la cuenta de Twitter de Podemos, se observó una tendencia lineal más o menos constante, que decrecía progresivamente conforme se acercaban las elecciones. Así, los días posteriores al acontecimiento se percibió cierto crecimiento en la puntuación final de sus seguidores, hecho que podía derivar de la polémica que suscita una formación política novedosa. Sin embargo, este crecimiento de la puntuación fue leve y sin valores atípicos, descartando con ello cualquier tipo de sospecha sobre este grupo de usuarios, ya que presentaban valores entendidos como normales y esperados en el crecimiento saludable de cualquier cuenta de la plataforma. Por tanto, estos usuarios fueron catalogados como "cuentas buenas".
- Finalmente, en el caso de los resultados vinculados al $P(CC_t)$ de los seguidores de la cuenta de Twitter de Vox, se contempló una tendencia lineal con un crecimiento muy notorio a partir del día de las elecciones. Este hecho podía derivar del debate que produce un partido nuevo, al igual que en el caso anterior. Sin embargo, este crecimiento importante llamaba la atención, ya que se alejaba de los valores considerados como normales para el crecimiento de cualquier cuenta de la plataforma. Así, este hecho llevaba a pensar que los resultados podían haberse visto influenciados por algún tipo de campaña o estrategia. Por tanto, estos usuarios fueron catalogados como "cuentas malas".

3.4. Conclusiones y trabajo pendiente

El algoritmo que se desarrolló se vio que cobraba solidez y efectividad si se aplicaba a un conjunto de datos recopilados durante un periodo prolongado, ya que aquello que realmente aportaba valor a la hora de identificar una cuenta como buena o mala, era la tendencia obtenida a partir de las puntuaciones finales de cada usuario a lo largo del tiempo. Así, con los datos que se obtuvieron, se pudo demostrar que el algoritmo ideado era válido como instrumento de clasificación, pero era mejorable. De esta forma, aplicando el algoritmo diseñado se conseguía de forma muy clara detectar aquellas cuentas que eran bots, usuarios falsos o que generaban

algún tipo de sospecha, aportando así una solución más en la lucha contra la detección de este tipo de perfiles. Sin embargo, en aquellas cuentas falsas que simulaban ser usuarios reales, el algoritmo no era capaz de detectarlas de forma clara, pudiendo estas pasar desapercibidas. Además, otro aspecto a destacar del algoritmo diseñado era su coste computacional lineal, pues únicamente dependía de la cantidad de registros de información que tuviera que procesar.

Así, tal y como se ha mencionado, se quedó trabajo por realizar, ya que existían diversos aspectos que podían mejorarse como las siguientes cuestiones:

- Sustitución de valores discretos por valores continuos, con el fin de perfeccionar la puntuación final obtenida para cada cuenta.
- Incluir una mayor cantidad de factores que influyan en el cálculo de la puntuación final, tanto determinantes como no determinantes, así como repensar si los actuales son relevantes o de lo contrario no resultan significativos y se puede prescindir de ellos.
- Terminar con los falsos positivos estudiando para ello el efecto que provocan las publicaciones que realiza un usuario según su calidad y repercusión.
- Analizar la periodicidad con la que se efectúan las publicaciones en una cuenta, con el fin de detectar posibles patrones sistemáticos que permitan identificar si se trata de publicaciones programadas en intervalos concretos de tiempo o, de lo contrario, están sujetas a la aleatoriedad humana.
- Observar el lenguaje utilizado en cada una de las publicaciones de una cuenta con el fin de analizar los contextos de sus palabras, así como la presencia de términos que expresen ideas o sentimientos, con el fin de afinar si se trata de una persona real o no.
- Crear un algoritmo de clasificación automático que permita recoger aspectos relevantes para la credibilidad como valores máximos y mínimos o la tendencia media, entre otros. Asimismo, podría apoyarse en el uso de técnicas como machine learning o redes neuronales que permitan aprender de la experiencia de clasificaciones previas.

4. Estado del arte

En este apartado se procede a revisar algunas investigaciones y trabajos relacionados con el presente proyecto. De esta forma, se han revisado varias propuestas enfocadas hacia distintas estrategias sobre cómo abordar el problema sujeto a estudio, así como las técnicas empleadas para el procesamiento de datasets o conjuntos de datos.

4.1. Detección de cuentas falsas en redes sociales

Hoy en día las redes sociales en línea (OSN – Online Social Network) se consideran centros de redes donde millones de usuarios comparten gustos, intereses y se relacionan entre sí a nivel mundial. Con el paso del tiempo, el uso de estas plataformas ha ido incrementándose, pero simultáneamente también lo han hecho los problemas de seguridad y privacidad derivados de las mismas, hecho que ha generado un aumento considerable de distintos tipos de cibercrímenes [10].

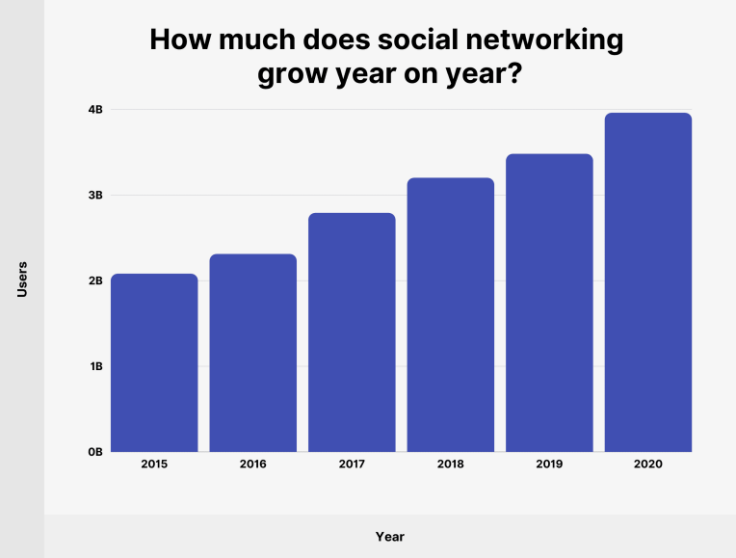


Ilustración 1. Evolución de usuarios activos en redes sociales desde 2017.
(Fuente: https://backlinko.com/)

Los usuarios en estas plataformas comparten gran cantidad de información que en muchas ocasiones se trata de datos personales, no siendo conscientes así de las amenazas o riesgos que ello puede implicar. Así, algunas de estas amenazas a las que se exponen resultan ser la generación de perfiles falsos y la clonación de perfiles. Por un lado, la creación de cuentas falsas supone un riesgo realmente importante en la seguridad de estas plataformas, pues simulan ser

personas o entidades inexistentes en la red social y cuyas intenciones suelen ser malignas en la mayoría de los casos como amenazar, desinformar o actividades de spam. Por otro lado, la clonación de perfiles consiste en el robo de datos de cuentas reales con el fin de generar perfiles repetidos con los que posteriormente llevar a cabo actividades maliciosas como pueden ser el acoso, la suplantación de identidad o la difusión de spam, entre otras. Asimismo, en esta clonación de perfiles se pueden distinguir dos circunstancias principales, bien que la clonación de una cuenta se realice para generar un nuevo perfil en la misma red social de origen (Clonación en la misma plataforma), o bien, que con la clonación de una cuenta se genere un nuevo perfil en otra red social donde el usuario dueño de esa información no dispone de cuenta (Clonación entre plataformas) [11].

Así pues, este tipo de perfiles no fiables se han convertido en una realidad muy presente hoy en día en las redes sociales y, por ello, es necesario encontrar mecanismos que permitan identificar este tipo de cuentas engañosas. De esta forma, existen numerosos estudios donde se han propuesto ya ideas muy diversas para colaborar con este hecho, algunos de los cuales se citan a continuación.

4.2. Detección basada en reglas

En primer lugar, cabe destacar una propuesta que consistía en **detectar tanto cuentas falsas como cuentas clonadas en la red social Twitter**, llevando a cabo dicha identificación de cada uno de estos tipos de perfiles mediante [12]:

- **Detección de cuentas falsas basada en reglas,** donde de forma efectiva conseguían distinguir los perfiles falsos de los reales. Para ello, analizaban una serie de reglas sobre cada una de las cuentas sujetas a estudio y se les iba asignando una puntuación según si cumplían o no con el criterio establecido.

  Así, algunas de las comprobaciones que se ejecutaron sobre las cuentas a estudiar fueron si disponían de nombre de perfil, imagen personalizada y descripción, si tenían la geolocalización activa, si publicaban grandes cantidades de tuits o si de lo contrario no publicaban nada, entre otras.

  De esta forma, con las puntuaciones obtenidas para cada criterio anterior se conformaba una puntuación final que, si superaba el valor del umbral prestablecido, se determinaba que la cuenta analizada era falsa.
- **Detección de cuentas clonadas utilizando medidas de similitud,** diferenciando entre similitud de atributos y similitud de red. Para ello, inicialmente extraían la información

personal del perfil del usuario considerado como “víctima” y, posteriormente, buscaban perfiles que presentasen datos coincidentes con la información extraída.

Por un lado, **la similitud de atributos** se calculaba en función del grado de semejanza existente entre los perfiles de la comparativa, siendo las propiedades a comparar el nombre, el nombre de pantalla, la localización geográfica, el idioma y la zona horaria. Esta puntuación se obtenía en función de la similitud del coseno y la distancia de Levenshtein.

Por otro lado, **la similitud de red** se calculaba en base a la semejanza de las relaciones de redes de contactos entre usuarios, analizando para ello el atributo “Followers_ids” que contenía el listado con las cuentas seguidoras del usuario involucrado en el análisis. Esta puntuación se obtenía aplicando su respectiva fórmula y, si la puntuación de similitud obtenida era mayor que el valor del umbral prestablecido se determinaba que la cuenta analizada era un clon.

- **Detección de perfiles clonados usando el algoritmo de árbol de decisión C4.5,** el cual construía el árbol en función de la información proporcionada con el fin de detectar posibles similitudes entre los atributos. De esta forma, comparaba el perfil dado como entrada con aquellos perfiles almacenados en base de datos y, en el caso de que existiera coincidencia, se determinaba que la cuenta analizada era un clon.

Estos métodos se aplicaron a un conjunto de muestras de usuarios reales y falsos recopilados de proyectos MIB, donde 2200 cuentas se destinaron a probar la detección de perfiles falsos (1100 de ellas falsas) y 780 fueron utilizadas para comprobar el funcionamiento de los métodos de detección de clones (20 de ellas clonadas). Así, tras aplicar las soluciones propuestas sobre los conjuntos de datos, se pudo concluir que, por un lado, **el método de detección de cuentas falsas era capaz de clasificar tanto cuentas falsas como verdaderas de forma correcta**. Por otro lado, tras comparar los resultados obtenidos con los **dos métodos utilizados para la detección de cuentas clonadas**, medidas de similitud y algoritmo C4.5, se vio que **la detección usando las medidas de similitud presentaba un mejor funcionamiento** y, además, era capaz de detectar la gran mayoría de clones que se introdujeron en el sistema.

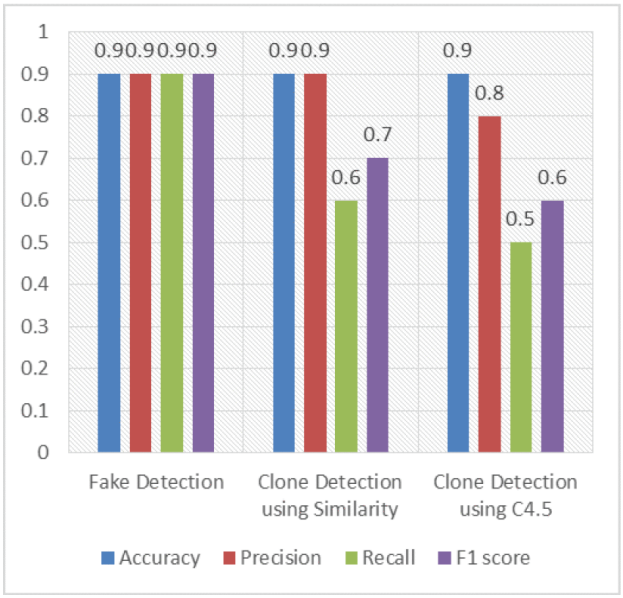


Ilustración 2. Resultados de la evaluación del desempeño para cada método de detección (Fuente https://ieeexplore.ieee.org/)

4.3. Detección utilizando Machine-Learning

En segundo lugar, se destaca una propuesta que consistía en **detectar en tiempo real perfiles y comportamientos maliciosos en Twitter utilizando para ello Machine-Learning** [13]. Para ello, se propuso un analizador basado en la detección automatizada de perfiles falsos mediante la identificación de características y comportamientos a través de una extensión para el navegador web Chrome. Tras llevar a cabo la evaluación de cada cuenta, se generaba una puntuación de confianza que permitía distinguir las cuentas falsas con una precisión muy alta. Este valor se obtenía mediante la fórmula de la puntuación de confianza o "true score" [14] que calculaba dicho valor para cada nodo mediante el análisis de atributos en redes sociales. De esta forma, si el valor obtenido no superaba el umbral establecido, se entendía que se trataba de una cuenta falsa.

Asimismo, la extracción de los datos de las cuentas se llevó a cabo mediante la API de Twitter y su rastreador web, recopilando tanto información pública como privada. Así, recopilaron información de más de 6500 cuentas de usuarios de la red social, entre las cuales existían perfiles con clara intención maliciosa.

De esta manera, con el análisis del conjunto de datos recopilado se pretendía extraer las características principales y el comportamiento de cada una de las cuentas seleccionadas, con el fin de clasificarlas como reales o falsas. Seguidamente, estos datos se trasladaban a una

plataforma de aprendizaje automático conocida con el nombre de WEKA y se procesaban utilizando técnicas de clasificación muy diversas como Random Forest, Bagging, JRip, J48 y PART.

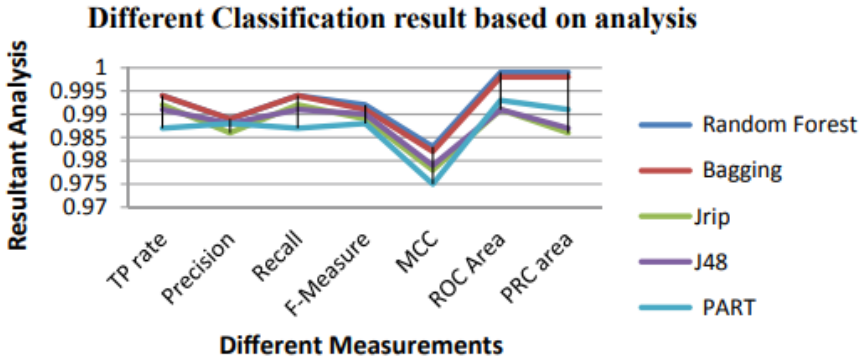


Ilustración 3. Resultados del análisis para cada clasificador
(Fuente: https://link.springer.com)

Finalmente, tras realizar el análisis, se observó que de entre todos los clasificadores aplicados sobre el conjunto de datos a catalogar, Random Forest y Bagging fueron los que ofrecieron resultados más prometedores en la clasificación de cuentas. Así pues, para aumentar el rendimiento del analizador, finalmente se optó por hacer uso de ambos para la detección de perfiles fraudulentos.

4.4. Detección utilizando NPL

En tercer lugar, cabe destacar una propuesta que consistía en **la identificación de perfiles falsos en la red social Twitter aplicando para ello técnicas de preprocesamiento del lenguaje natural, reducción dimensional y machine learning** [15]. Así pues, para ello, se distinguieron 5 pasos principales en este proyecto:

- **Recopilación de datos correspondientes a perfiles reales y falsos,** uno de los puntos más relevantes. En él, se reunían todas las características o aspectos de los perfiles para su estudio y evaluación.
- **Preprocesamiento de la información utilizando técnicas NLP (Natural Language Processing),** mediante las cuales se reducía la cantidad de información recopilada con el fin de mejorar el rendimiento. Así, las técnicas que se emplearon fueron **extracción, eliminación de palabras vacías** o redundantes, **stemming** para la obtención de las raíces de las palabras y **tokenización** para dividir textos en oraciones y posteriormente en palabras.
- **Reducción dimensional aplicando la técnica PCA (Principal Component Analysis),** obteniendo con ello únicamente las variables más relevantes y significativas de entre

todas las características recopiladas de cada perfil. En este caso, las variables que se consideraron óptimas fueron 6: cantidad de seguidores, cantidad de cuentas a las que sigue el perfil, cantidad de tweets y retweets totales emitidos por el usuario, código de lenguaje, cantidad total de favoritos dados por el usuario a otras publicaciones y número de listas a las que pertenece la cuenta.

- **Clasificación de los perfiles** mediante las técnicas de machine learning **Naïve Bayes, SVM y ISVM**, con el fin de posteriormente comparar la precisión obtenida en las clasificaciones realizadas por los 3 modelos y determinar cuál arroja una mayor precisión en esta tarea.

Table 1. Classification accuracy of the three models, SVM, ISVM and NB

	SVM	Naïve Bayes	ISVM
Accuracy	0.774	0.773	0.900

Ilustración 4. Precisión obtenida en la clasificación para cada modelo (Fuente: https://www.journaljamcs.com/)

- **Evaluación de los resultados obtenidos** para cada modelo**,** analizando para ello aspectos relacionados con la precisión y la eficiencia conseguidas en la clasificación.

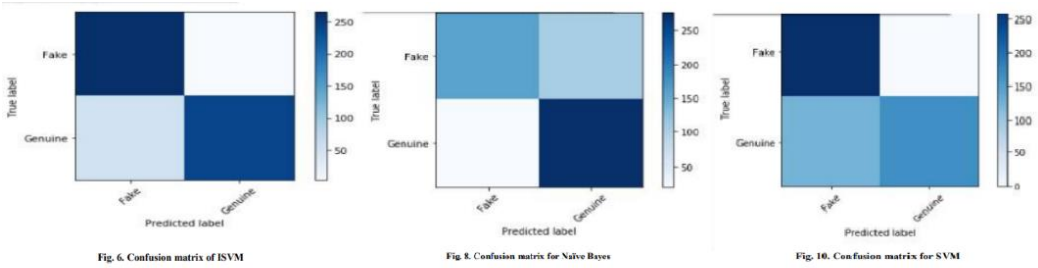


Fig. 6. Confusion matrix of ISVM

Fig. 8. Confusion matrix for Naïve Bayes

Fig. 10. Confusion matrix for SVM

Ilustración 5. Matrices de confusión obtenidas para los 3 modelos (Fuente: https://www.journaljamcs.com/)

Esta propuesta planteada se aplicó a un conjunto de más de 100.000 registros de usuarios de Twitter pertenecientes a unos 37 países distintos. Tras pasar estos perfiles por los modelos de aprendizaje, se procedió a clasificarlos mediante los algoritmos anteriormente indicados, logrando la mayor precisión de un 90% con el modelo ISVM. Así, gracias al uso de las técnicas ISVM y PCA se consiguieron mejores resultados en la detección de este tipo de perfiles falsos en redes sociales.

4.5. Detección a través de características

En cuarto lugar, se destaca una propuesta que consistía en **identificar cuentas falsas utilizando un enfoque basado en características de los perfiles** y, posteriormente, **evaluar la precisión del**

modelo mediante 3 algoritmos distintos de machine learning: Random Forest, SVM y Logistic Regression [16]. Para ello, en primer lugar, se utilizaron conjuntos de datos disponibles de forma pública obtenidos a través de la API de Twitter tanto de cuentas de usuarios como de publicaciones. En este caso utilizaron para el estudio más de 11.000 cuentas de usuarios pertenecientes a perfiles reales, falsos y de spam. Seguidamente, estos datos se preprocesaron para eliminar la información irrelevante y, tras ello, se llevó a cabo la selección de 24 características basadas en aspectos de las cuentas, publicaciones, detalles de propiedad y en la presencia de enlaces en los tuits. Posteriormente, se procedió a entrenar a los modelos de clasificación con el conjunto de registros recopilado de la red social, distinguiendo entre datos para entrenamiento, con los que ajustar los parámetros del clasificador y datos para pruebas, empleados para afinar la arquitectura del clasificador. Así, una vez se entrenó el clasificador, se efectuó la clasificación de los perfiles en reales y falsos, siendo posteriormente evaluada la precisión obtenida por los algoritmos de machine learning seleccionados.

Table 3. Shows Results of Supervised Machine Learning Algorithm

Model	W-Accuracy	Precision	Recall	F-Score
Logistic Regression	95.7%	94	96	95
SVM	80.8 %	82	100	90
Random Forest	97.9%	98	98	98

Ilustración 6. Evaluación y resultados de las métricas para cada modelo (Fuente: https://dl.acm.org/)

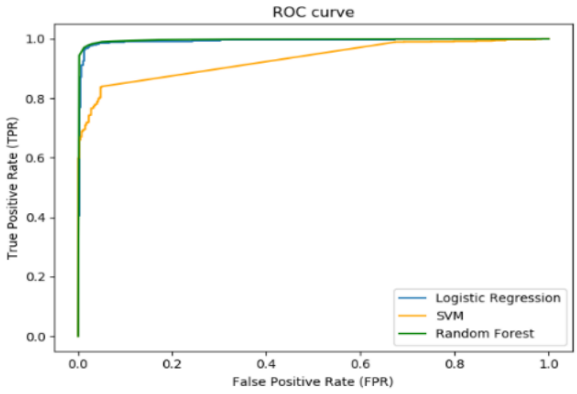


Figure 2. ROC Curve for different models

Ilustración 7. Curva ROC obtenida para cada uno de los modelos (Fuente: https://dl.acm.org/)

Así, tras evaluar los resultados obtenidos para esta propuesta centrada en la detección de perfiles falsos mediante características, se observó que el algoritmo Random Forest (97.9%) lograba mejores resultados en la clasificación al presentar una mayor precisión y eficiencia que los modelos Logistic Regressión (95.7%) y SVM (80.8%). Además, se observó que las predicciones

fallidas eran mayoritariamente casos de perfiles falsos manejados por personas reales, hecho que dificultaba mucho su correcta identificación.

4.6. Detección utilizando ontologías

En quinto lugar, cabe destacar una propuesta que consistía en **identificar cuentas falsas en Twitter utilizando ingeniería ontológica y reglas SWRL (Semantic Web Rule Language)** [17]. Así pues, en este estudio se distinguieron 3 etapas principales:

- **Preprocesamiento de datos y selección de características,** utilizando para ello un conjunto de datos ya existente publicado por el Instituto de Informática y Telemática del consejo nacional de investigación de Italia (IIT-CNR), el cual contenía miles de cuentas de usuarios y millones de publicaciones. Seguidamente, se preprocesó dicha información con el fin de eliminar aquellos datos incorrectos, incompletos, inconsistentes y no relevantes, aplicando también conversiones de tipos en aquellos datos donde fue necesario. Además, se analizaron las características disponibles de cada una de las cuentas con el fin de escoger aquellas que fueran de mayor relevancia para el estudio.
- **Construcción de ontología**, proponiendo en este caso la ontología denominada FADCO (Fake Account Detection and Classication Ontology). De esta manera, su arquitectura se componía de los siguientes pasos principales: determinar el dominio y alcance de la ontología, definir las clases, sus jerarquías y propiedades de clases, así como crear las instancias y escribir las reglas SWRL de fisionomía. Así pues, en este caso se definieron como clases principales "Cuenta" y "Persona", entendiendo que una persona podía disponer de un objeto cuenta. En cuanto a propiedades de las clases, se definieron mayoritariamente en la clase "Cuenta", destacando algunas de especial relevancia como el número de seguidores o el número de amigos, entre otras.

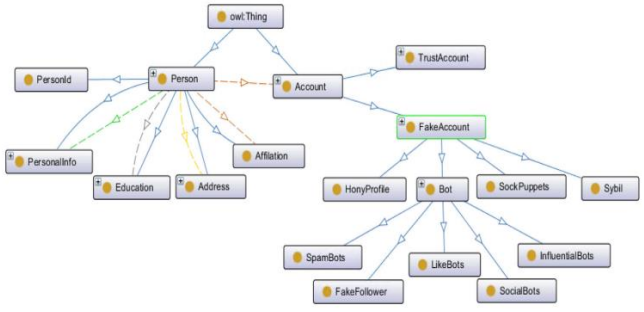


Fig. 2. The classes hierarchy of Fake Account Detection and Classification Ontology (FADCO).

Ilustración 8. Jerarquía de clases propuesta para la ontología FADCO (Fuente: https://kijoms.uokerbala.edu.ig/)

- **Clasificación con razonador y reglas SWRL (Semantic Web Rule Language),** donde se aplicó el razonador Pellet ya que presentaba una utilidad directa para trabajar con reglas OWL y SWRL, permitiendo además definir reglas SWRL de manera personalizada. En cuanto a las reglas semánticas, estas se establecieron en función de relaciones fuertes para la detección de cuentas falsas utilizando las características extraídas de los perfiles. Así, para la clasificación multi-etiqueta utilizaron un enfoque que consistía en estimar la probabilidad de pertenencia de un objeto a una clase determinada, comprobando mediante un conjunto de reglas si la instancia cumplía o no las condiciones establecidas para ser miembro.

Esta propuesta planteada se aplicó a un conjunto de datos que contenía más de 11.000 cuentas de Twitter, tanto reales como falsas, junto con más de 12 millones de publicaciones. Así, tras efectuar la clasificación con el método propuesto, se evaluó su precisión con respecto a la obtenida con otros algoritmos de machine learning.

Table 8
Evaluation metrics show the accuracy of an ontology vs some machine learning techniques.

Technique	Recall	Precision	F-Measure	Accuracy
NAIVE-BAYS	0.990	0.930	0.959	0.945
Ontology	**0.975**	**0.986**	**0.980**	**0.972**
Logistic	0.981	0.987	0.984	0.978
SVM	0.984	0.987	0.985	0.979

Ilustración 9. Comparativa de precisión entre ontología y técnicas de machine learning (Fuente: https://kijoms.uokerbala.edu.iq/)

De esta manera, se obtuvo una precisión para el método ontológico del 97.5%, hecho que evidenciaba su gran potencial, pues presentaba valores muy similares al resto de algoritmos e incluso los mejoraba en algunas de las métricas. Además, este enfoque ontológico resultó ser una solución mucho más interpretable y comprensible, al permitir establecer reglas de decisión de forma rápida y poder conocer los razonamientos que han llevado al proceso a tomar unas decisiones u otras.

4.7. Detección mediante aprendizaje semi-supervisado

En sexto lugar, se destaca una propuesta que consistía en **identificar perfiles falsos en Twitter mediante un algoritmo de aprendizaje semi-supervisado basado en gráficos (EGSLA)** [18]. Para ello, el método propuesto constaba de 3 fases principales, las cuales resultaban ser las siguientes:

- **Recopilación de datos,** donde se extrajo contenido generado por usuarios (UGC) de la red social Twitter mediante un framework para Python de web-scraping. Posteriormente, estos datos se almacenaron en una base de datos orientada a grafos, Neo4J. En este caso se recopilaron datos de Twitter durante 2 meses, realizando especial hincapié en coleccionar publicaciones que tuvieran enlaces o temas/etiquetas en tendencia conocidas como "trending topics", en intervalos de 1 hora.
- **Extracción de características,** llevando a cabo este paso mediante el preprocesamiento de los datos obtenidos utilizando Apache Spark con el fin de darles formato y estandarizarlos, pudiendo crear así las colecciones de datos. Así, las características seleccionadas resultaron ser las siguientes: identificador del usuario, estado, lista de seguidores, información de seguimiento, fracción de URLs, tiempo de publicación, fracción de retuits, longitud media de los tuits y tiempo medio transcurrido entre tuits.
- **Clasificación y toma de decisiones,** donde se emplearon las características extraídas anteriormente, ya que el enfoque basado en gráficos depende tanto de la actividad de los usuarios como de los detalles de sus cuentas.

Esta propuesta planteada se evaluó empleando publicaciones reales de la red social Twitter, utilizando casi 3 millones de tuits procedentes de más de 20.000 usuarios distintos. Así, para valorar la actuación del modelo propuesto se hizo una comparativa con otros algoritmos de machine learning utilizados con frecuencia en la detección de cuentas falsas en redes sociales: teoría de juego, KNN, SVM y árbol de decisión.

Table 9 Accuracy in detecting fake users

Percentage of labeled data (%)	EGSLA	Game theory	KNN ($k=10$)	SVM	Decision tree
8	63.5	58.2	43.1	41.2	42.1
10	70.9	67.1	44.2	43.0	43.2
15	78.5	72.0	57.2	54.1	54.8
20	84.4	74.9	61.1	62.1	61.2
30	90.3	75.1	63.0	63.2	63.1

Ilustración 10. Precisión obtenida en la detección de perfiles falsos para cada modelo (Fuente: https://link.springer.com/)

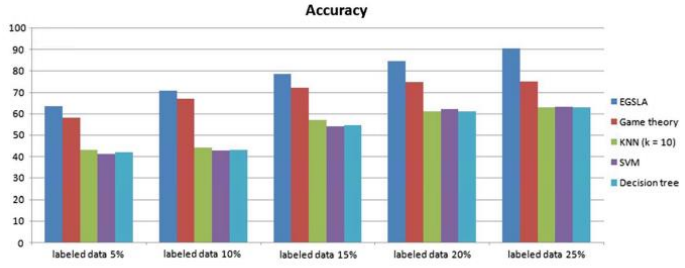


Fig. 8 Comparison of accuracy of fake user detection algorithms

Ilustración 11. Comparativa de precisión obtenida en los distintos algoritmos (Fuente: https://link.springer.com/)

De esta forma, tras comparar los resultados obtenidos para cada algoritmo, se observó que el método de detección propuesto EGSLA superaba en todas las métricas al resto de algoritmos con una precisión del 90.3%. Así, se consiguió demostrar en el estudio que el modelo propuesto EGSLA lograba distinguir de forma clara las cuentas falsas dentro del conjunto de usuarios de Twitter.

5. Objetivos

Este proyecto presenta varios objetivos, destacando entre ellos, como meta principal, la experimentación con técnicas de machine learning en la búsqueda de soluciones que colaboren en la detección de cuentas falsas en las redes sociales, enfocando especialmente este trabajo sobre la plataforma Twitter. De esta forma, se pretende seguir con la línea de investigación ya iniciada con anterioridad en mi Trabajo Final de Grado (TFM) apostando por tecnologías y soluciones más sofisticadas, como el uso de Inteligencia Artificial.

Además, se estudiará la relevancia que presenta cada una de las características presentes en los perfiles de usuarios que se hayan recopilado, con el fin de revisar su grado de influencia en la detección de cuentas falsas y poder así determinar si es o no decisiva en la clasificación de los perfiles.

Así, se conformará un conjunto de datos de cuentas de usuarios de Twitter, tanto perfiles reales como falsos, mediante datasets existentes y nuevos aportes propios. Este dataset servirá posteriormente para conformar los grupos de datos de entrenamiento y de pruebas para realizar una experimentación sobre distintos algoritmos de detección con el fin de ver cuál de ellos y qué configuración arrojan mejores resultados y permiten efectuar la clasificación de cuentas reales y falsas obteniendo una precisión óptima.

Asimismo, se destacan algunos objetivos secundarios o subobjetivos y se procede a detallarlos a continuación:

- Determinar las características más relevantes del listado completo que ofrece la API de Twitter para el objeto usuario, con el fin de prescindir de aquella información que no sea decisiva o influyente en la detección de cuentas falsas y priorizar aquellos aspectos que sí sean relevantes en dicha tarea.
- Proporcionar una solución que mejore la propuesta realizada con anterioridad en mi Trabajo de Fin de Grado mediante el uso de soluciones más sofisticadas como resultan ser el machine learning o el deep learning.
- Aprender y adquirir conocimientos sobre el mundo de la Inteligencia Artificial, haciendo especial hincapié en las disciplinas científicas de machine learning y deep learning, así como estudiar, experimentar y adquirir experiencia con tecnologías actuales muy presentes y utilizadas en este campo como es el caso de TensorFlow.

6. Preparación del entorno de trabajo

6.1. Equipos empleados en el desarrollo

En este apartado se procede a indicar con detalle las características principales de los equipos implicados en el desarrollo. Puesto que una de las partes más críticas del proyecto consiste en extraer información de la API de Twitter y cuya duración puede prolongarse varios días, para evitar posibles pérdidas de datos ante cualquier imprevisto como cortes de luz, desconexiones de internet o fallos del sistema, entre otros, se propone ejecutar el proceso de extracción de datos en paralelo sobre 2 equipos. De esta forma, en el caso de que el equipo principal fallase siempre se dispondría de la recopilación de datos realizada por el secundario como garantía.

Así pues, las características principales de ambos equipos portátiles empleados en el desarrollo resultan ser las siguientes:

Tabla 4. Características principales de los equipos portátiles empleados

Técnica de selección	Selección automática	Cantidad total de características
Fabricante del sistema	ASUSTeK COMPUTER INC	ASUSTeK COMPUTER INC
Modelo del sistema	ZenBook Pro 15 UX550GDX_UX580GD	N750JK
BIOS	UX550GDX.312	N750JK.204
Procesador	Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz (12 CPUs)	Intel(R) Core(TM) i7-4700HQ CPU @ 2.40 GHz (8 CPUs)
Memoria RAM	16,0 GB	16,0 GB
Sistema operativo	Windows 10 Pro 64 bits	Windows 10 Home 64 bits

6.2. Máquina virtual con Ubuntu

En primer lugar, se procede a montar un entorno virtualizado con el fin de disponer de un escenario limpio sobre el cual poder llevar a cabo el desarrollo de este proyecto. Para ello, se ha optado por escoger el sistema operativo Ubuntu, pues resulta ser una opción de código abierto muy ligera para su ejecución y una de las opciones favoritas para la comunidad de desarrolladores experimentados debido a las facilidades que ofrece para las labores de desarrollo de entre las que destaca su amplia variedad de herramientas y bibliotecas, entre otras [19].

Así pues, para la realización de este proyecto se ha optado por la versión estable más reciente de este sistema operativo, tal y como se indica en su página oficial desde donde podremos también descargarla, la "20.04.2.0 LTS".

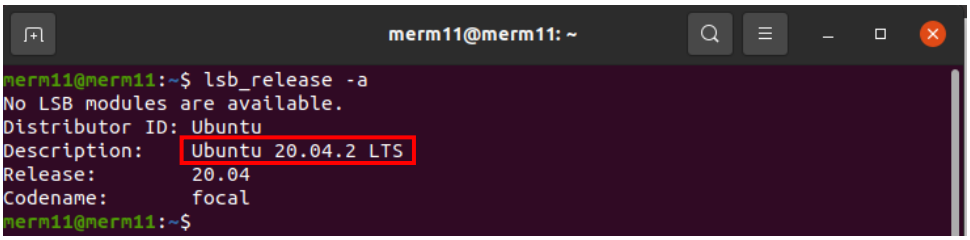


Ilustración 12. Captura de pantalla con la versión de Ubuntu instalada (Fuente propia)

6.3. Python y PyCharm

Para el desarrollo de este proyecto, puesto que se parte originalmente de una investigación propia anterior, se ha apostado de nuevo por el lenguaje de programación Python esta vez en su versión "3.8.10". Es cierto que existen versiones más modernas, sin embargo, se ha escogido esta debido a su buena compatibilidad con otras herramientas y librerías que se utilizarán más adelante en el proyecto.

Así, en primer lugar, para poder comenzar con su instalación [20] deberemos actualizar el listado de paquetes del sistema operativo e instalar algunos requisitos necesarios mediante los siguientes comandos:

```
$ sudo apt-get update
```

```
$ sudo apt install software-properties-common
```

Seguidamente, será preciso agregar el siguiente repositorio al listado del sistema:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
```

Una vez habilitado el repositorio, ya será posible proceder con la instalación de Python y verificar posteriormente la misma mediante los siguientes comandos:

```
$ sudo apt install python3.8
```

```
$ python3 -V
```

Así, al finalizar la instalación, el sistema devolverá la versión recién instalada de Python, tal y como puede observarse en la siguiente captura. En este caso se dispone de la versión "3.8.10".

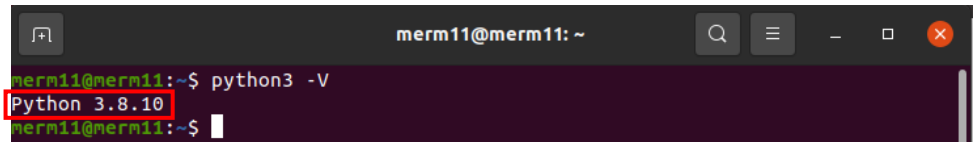


Ilustración 13. Captura de pantalla mostrando la versión de Python instalada
(Fuente propia)

Seguidamente, se procederá a descargar e instalar un entorno de desarrollo integrado (IDE) con el que poder desarrollar y ejecutar proyectos con el lenguaje escogido Python. Para ello, la herramienta seleccionada resulta ser PyCharm en su versión "Community Edition 2021.2", pues es gratuita y se considera el mejor IDE para llevar a cabo el desarrollo de proyectos con este lenguaje [21].

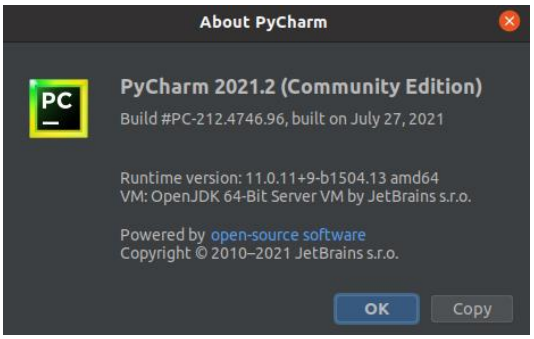


Ilustración 14. Captura de pantalla mostrando la versión de PyCharm instalada
(Fuente propia)

Así pues, esta herramienta se obtiene descargándola desde su página web oficial [22]. Seguidamente, una vez instalada deberemos ejecutarla y, antes de crear cualquier proyecto, será necesario configurar qué intérprete de Python deberá utilizar el IDE, seleccionando en este caso Python 3.8, tal y como puede observarse en la siguiente captura.

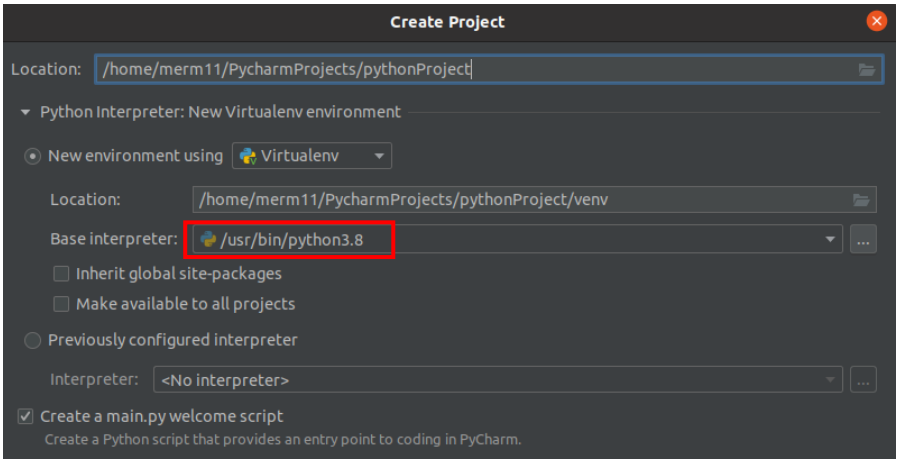


Ilustración 15. Captura de pantalla sobre la configuración del intérprete de Python en Pycharm
(Fuente propia)

6.4. Librería Tweepy

Para poder llevar a cabo la extracción de características de cada uno de los perfiles de usuario, será necesario hacer uso de la librería Tweepy [23]. Esta permitirá establecer la conexión con la API de Twitter y poder así llevar a cabo la recogida de información de cada una de las cuentas.

Para ello, inicialmente, se deberá instalar el gestor de paquetes de Python denominado "Pip", el cual permitirá añadir la librería requerida para la extracción: Tweepy. De esta manera, será necesario ejecutar el siguiente comando:

```
$ sudo apt install python3-pip
```

Así, una vez finalizada la instalación, el sistema devolverá la versión recién instalada de "Pip", tal y como puede observarse en la siguiente captura. En este caso se dispone de la versión "20.0.2".

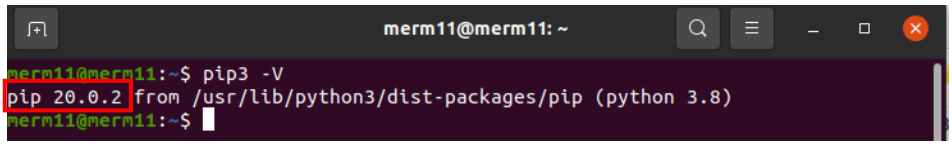


Ilustración 16. Captura de pantalla mostrando la versión de Pip instalada (Fuente propia)

Una vez ya instalado el gestor de paquetes, se podrá instalar la dependencia necesaria para la extracción de datos de la API, en este caso la librería Tweepy, mediante el siguiente comando:

```
$ pip3 install tweepy
```

De esta forma, tras la instalación podremos consultar la versión que se ha instalado de "Tweepy" mediante el comando "pip3 show", tal y como puede observarse en la siguiente captura. En este caso se dispone de la versión "3.10.0".

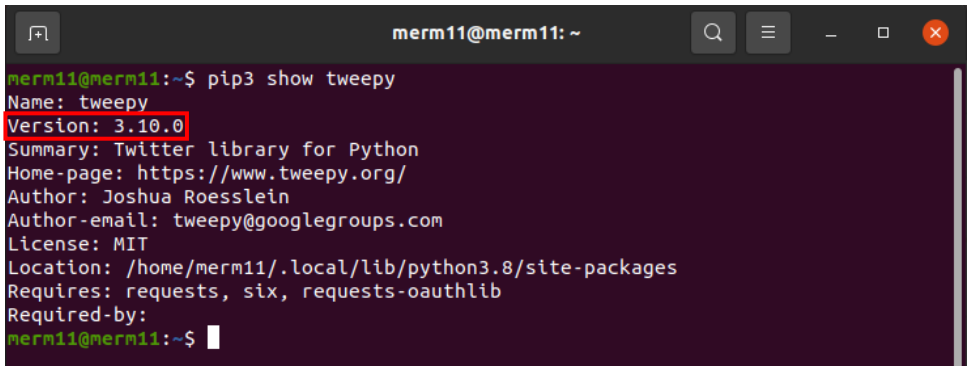


Ilustración 17. Captura de pantalla mostrando la versión de Tweepy instalada (Fuente propia)

6.5. Librería Scikit-learn

Para poder llevar a cabo codificaciones de datos y el uso de determinadas funciones de Machine learning en futuros apartados, será necesario hacer uso de la librería Scikit-learn, también conocida como "sklearn". Esta resulta ser una de las bibliotecas más prácticas y valiosas para tareas de Machine Learning en lenguaje Python ya que proporciona varios conjuntos de datasets, funciones como validación cruzada, algoritmos de aprendizaje supervisado y no supervisado, así como extracción y selección de características, entre otros aspectos [2424].

Puesto que ya se tiene instalado el gestor de paquetes, se podrá instalar la dependencia necesaria, en este caso la librería Scikit-learn, mediante el siguiente comando:

```
$ pip3 install scikit-learn
```

De esta forma, tras la instalación podremos consultar la versión que se ha instalado de "Scikit-learn" mediante el comando "pip3 show", tal y como puede observarse en la siguiente captura. En este caso se dispone de la versión "0.24.2".

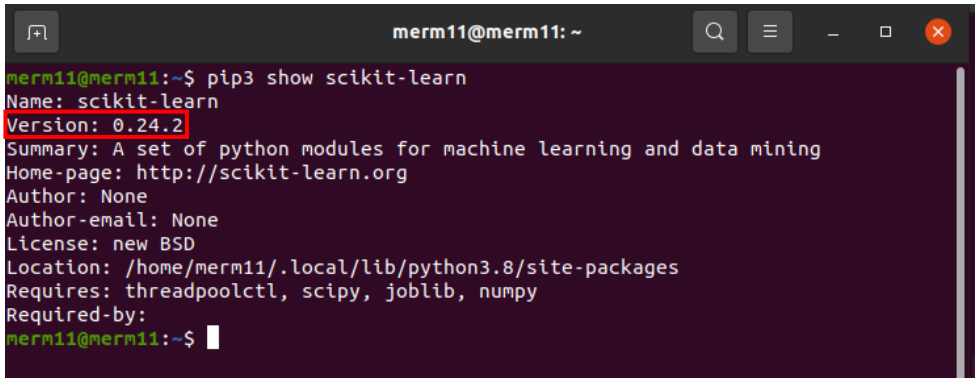


Ilustración 18. Captura de pantalla mostrando la versión de Scikit-learn instalada (Fuente propia)

6.6. Librería TensorFlow

Para poder llevar a cabo el entrenamiento de los modelos y las pruebas de clasificación de los datos de usuarios, será necesario hacer uso de la librería TensorFlow. Esta, tal y como se indica en su página web oficial, resulta ser la biblioteca de código abierto más relevante para llevar a cabo desarrollos y entrenamientos de modelos de Machine Learning. Para ello, proporciona una gran cantidad de herramientas, recursos y bibliotecas, con el fin de fomentar y facilitar tanto la investigación como el desarrollo de aplicaciones que empleen técnicas de Machine Learning [25].

Puesto que ya se tiene instalado el gestor de paquetes, se podrá instalar la dependencia necesaria, en este caso la librería TensorFlow, mediante el siguiente comando:

```
$ pip3 install tensorflow
```

De esta forma, tras la instalación podremos consultar la versión que se ha instalado de "TensorFlow" mediante el comando "pip3 show", tal y como puede observarse en la siguiente captura. En este caso se dispone de la versión "2.4.1".

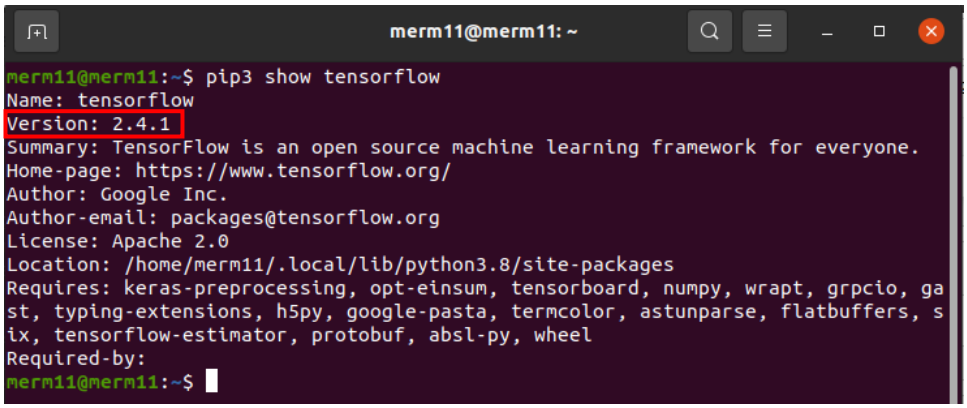


Ilustración 19. Captura de pantalla mostrando la versión de TensorFlow instalada (Fuente propia)

6.7. Pentaho Data Integration

En este apartado, se procede a detallar el proceso de instalación de Pentaho Data Integration o PDI, una herramienta de código abierto destinada a la integración de datos y especialmente utilizada para tareas relacionadas con la inteligencia de negocio o BI [26]. Así, esta herramienta se utilizará en este proyecto para diversas tareas de tratamiento de datos como la unificación de datasets o la eliminación de datos duplicados, entre otras.

Para comenzar su instalación, será necesario disponer de Java en su versión "1.8", ya que PDI requiere de esta versión. Por tanto, se debe consultar en primer lugar la versión de Java instalada en el equipo y, en el caso de que se disponga de una versión distinta a la necesaria, será necesario eliminarla.

Una vez hecho esto, para instalar Java 1.8 y configurarlo como la versión predeterminada del sistema se deberán ejecutar las siguientes líneas de comandos:

```
$ sudo apt install openjdk-8-jdk
```

```
$sudo apt install default-jre
```

De esta forma, tras finalizar la instalación el sistema devolverá la versión que se ha instalado de Java, tal y como se puede apreciar en la siguiente captura proporcionada, donde indica que se dispone de la versión “1.8”.

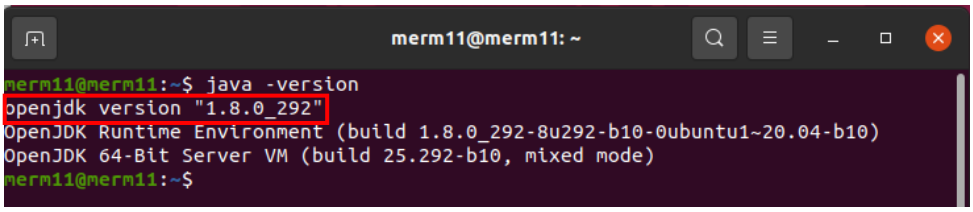


Ilustración 20. Captura de pantalla mostrando la versión de Java instalada (Fuente propia)

Tras instalar Java, se debe descargar la herramienta Pentaho en su última versión más estable desde su portal de descarga oficial [27], en este caso la versión “9.2”. Así pues, finalizada la descarga, se deberá extraer el fichero comprimido y acceder al directorio mediante terminal. De esta forma, una vez se esté en la ruta correspondiente, para lanzar finalmente la aplicación PDI se deberá ejecutar el archivo “spoon.sh” contenido en la carpeta “data-integration” utilizando para ello el siguiente comando:

```
$ sh spoon.sh
```

Seguidamente, se abrirá la aplicación mostrando la versión que se esté ejecutando en una ventana emergente, como se puede apreciar en la siguiente captura.

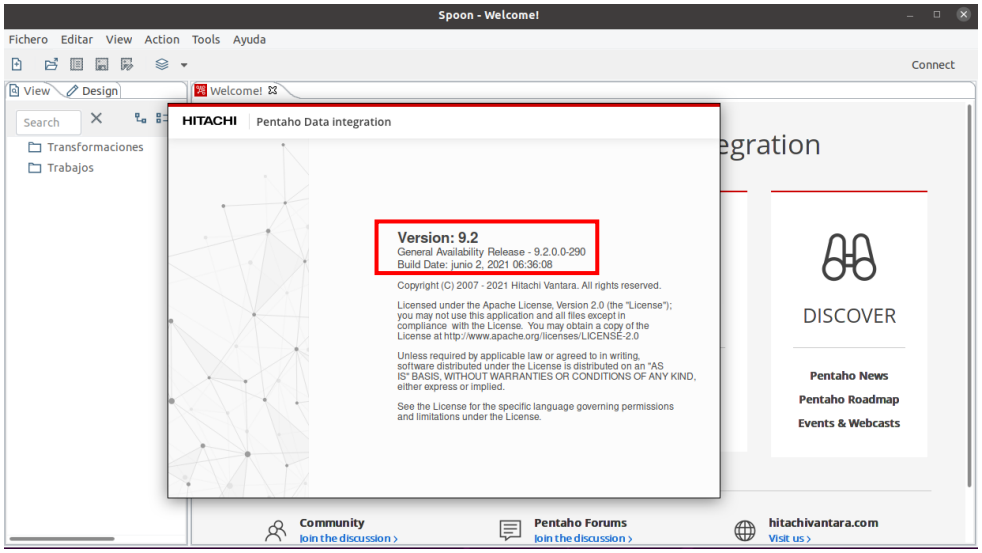


Ilustración 21. Captura de pantalla mostrando la herramienta ejecutada Pentaho (Fuente propia)

6.8. Anaconda y Jupyter Notebook

En este apartado, se procede a detallar el proceso de instalación de la distribución Anaconda para poder posteriormente ejecutar la herramienta Jupyter Notebook, la cual resulta ser una aplicación web de código abierto que permite la creación de documentos, así como ejecutarlos y compartirlos en vivo de una manera sencilla e interactiva. Esta herramienta es ampliamente utilizada para tareas como el procesamiento de datos, simulaciones matemáticas labores estadísticas, machine learning o representaciones gráficas de datos, entre otras. Además, la aplicación soporta más de 40 lenguajes distintos de programación, entre los cuales se destacan Julia, Python y R [28].

Así, esta herramienta se utilizará en este proyecto para diversas labores como el tratamiento de datos, el entrenamiento de modelos y la elaboración de representaciones gráficas, entre otras.

De esta forma, para poder hacer uso de esta herramienta será necesario descargar la distribución de Anaconda desde su sitio web oficial [29], donde **descargaremos la edición individual en su versión para Linux compatible con Python en su versión "3.8".** Una vez descargada, para que comience la instalación deberemos ejecutar el fichero ".sh" mediante el siguiente comando:

```
$bash FicheroInstalacionAnaconda.sh
```

Seguidamente, tras finalizar la instalación, podremos consultar la versión que se ha instalado de Anaconda, tal y como se puede apreciar en la siguiente captura proporcionada, donde indica que se dispone de la versión "4.10.3".

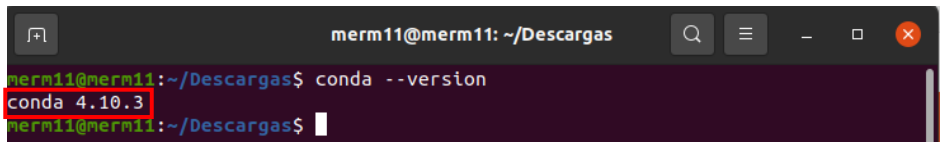


Ilustración 22. Captura de pantalla mostrando la versión de Anaconda instalada (Fuente propia)

Así, una vez instalada la distribución, se procede a crear un entorno de trabajo mediante la terminal, indicando que se desea instalar Jupyter notebook junto con el listado de librerías necesarias, quedando el comando de la siguiente manera:

```
$conda create -n entornotensorflow python=3.8 tensorflow=2 jupyter numpy
pandas scikit-learn matplotlib
```

De esta forma, comenzará la instalación de todos los paquetes necesarios dentro del entorno que hemos creado. Así, una vez finalizada la instalación, deberemos activar el entorno de trabajo mediante el siguiente comando:

```
$conda activate entornotensorflow
```

Una vez ya esté el entorno de trabajo activo en nuestra terminal, podremos consultar la versión que se ha instalado de Jupyter notebook, tal y como se puede apreciar en la siguiente captura proporcionada, donde indica que se dispone de la versión "6.4.3".

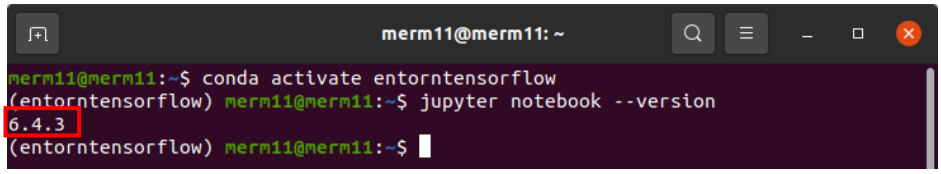


Ilustración 23. Captura de pantalla mostrando la versión de Jupyter notebook instalada (Fuente propia)

Así, una vez verificada la correcta instalación y su versión, para ejecutar finalmente Jupyter notebook deberemos utilizar el siguiente comando que abrirá directamente una página en nuestro navegador web con la herramienta funcionando:

```
$jupyter notebook
```

6.9. Resumen de inventario software

En este apartado se procede a listar todo el inventario del software más relevante desplegado en el entorno de trabajo, así como el propósito de cada uno de los elementos, pues todos ellos resultan necesarios o útiles para un fin particular.

Así pues, los programas o librerías empleados en el desarrollo de este proyecto resultan ser los siguientes:

Tabla 5. Inventario de software desplegado en el entorno de trabajo

Software	Versión	Propósito
Python	3.8	Lenguaje de programación empleado en el desarrollo del proyecto.
Pip	20.0.2	Administración de paquetes de software de Python.
Pycharm	Community Edition 2021.2	IDE empleado en el desarrollo con Python para la extracción de datos de usuarios de la API de Twitter y la selección de características.
Tweepy	3.10.0	Librería empleada para conectar con la API de Twitter y extraer la información de las cuentas de usuario.
Scikit-learn	0.24.2	Librería empleada para la codificación de datos y el uso de determinadas funciones de Machine learning.
TensorFlow	2.4.1	Librería empleada para el entrenamiento de los modelos y las pruebas de clasificación o predicciones.

Java	1.8	Requisito de instalación para Pentaho Data Integration.
Pentaho Data Integration	9.2	Unificación de datasets y eliminación de registros duplicados.
Anaconda	Individual Edition 4.10.3	Distribución que posibilita la creación de entornos de trabajo bajo unas determinadas dependencias.
Numpy	1.21.2	Librería empleada para convertir listas en arrays.
Pandas	1.3.3	Librería empleada para importar ficheros CSV.
Matplotlib	3.4.3	Librería empleada para pintar las gráficas correspondientes a la evolución de la precisión y la pérdida en los entrenamientos de los modelos.
Jupyter Notebook	6.4.3	Entorno web empleado para la visualización de las representaciones gráficas sobre la estructura de los modelos definidos, la evolución de la precisión y la pérdida en los entrenamientos de los modelos.

7. Elaboración del DataSet de experimentación

Aquí habría que decir que antes de desarrollar técnicas, es necesario poseer un dataset sobre el que realizar el trabajo y que con ese objetivo se va a construir uno a partir de datasets de otras comunidades pero además añadiendo tu registros en el dataset que ayuden al control certero de los datos, de esta forma evitas que igual la info de los datasets públicos que vas a utilizar como referencia si están contaminados pues por lo menos con tus registros de control puedes comprobar que todo va bien. Y ahora explicas como has creado el dataset.

En este apartado se procede a explicar con detalle cómo se ha llevado a cabo la elaboración del dataset o conjunto de datos a utilizar en la parte de experimentación de este proyecto. La composición de este fichero a utilizar vendrá dada por un listado de usuarios de la red social Twitter, sus características y su correspondiente etiquetado referente al tipo de cuenta en cada caso (bot o humano), así como la fecha y hora en la que se extrajo cada uno de los registros.

	A	B	C	D	E	F	G	H
1	fecha_hora_medicion	id	caracteristica1	caracteristica2	caracteristica3	...	caracteristicaN	tipo
2	2021-08-10 12:30:00	1	dato	dato	dato	...	dato	bot
3	2021-08-10 12:31:00	2	dato	dato	dato	...	dato	human
4	2021-08-10 12:32:00	3	dato	dato	dato	...	dato	human
5	2021-08-10 12:33:00	4	dato	dato	dato	...	dato	bot
6								

Ilustración 24. Ejemplo de fichero de salida esperado
(Fuente propia)

Así pues, inicialmente se partirá de varios datasets con listados de identificadores de usuarios de Twitter y su etiquetado, entre otros datos. De esta forma, una vez se unifiquen conservando únicamente identificadores y etiquetas de cada registro, será momento de solicitar a la API las características de cada perfil con el fin de extraerlas y posteriormente almacenarlas, elaborando con ello el dataset.

7.1. DataSet de elaboración propia

Inicialmente, se realizó una búsqueda manual por la red social Twitter con el fin de recopilar un listado de usuarios, tanto reales como falsos. De esta forma, se pretendía conformar un dataset de origen propio conociendo de antemano su etiquetado (bot o humano), a fin de disponer así de un conjunto de datos conocido y controlado.

De esta manera, se recopilaron un total de 210 usuarios conocidos de la plataforma donde se distinguían un total de 93 cuentas falsas o controladas por bots y 117 manejadas por personas o entidades reales, tal y como puede observarse en las siguientes capturas proporcionadas.

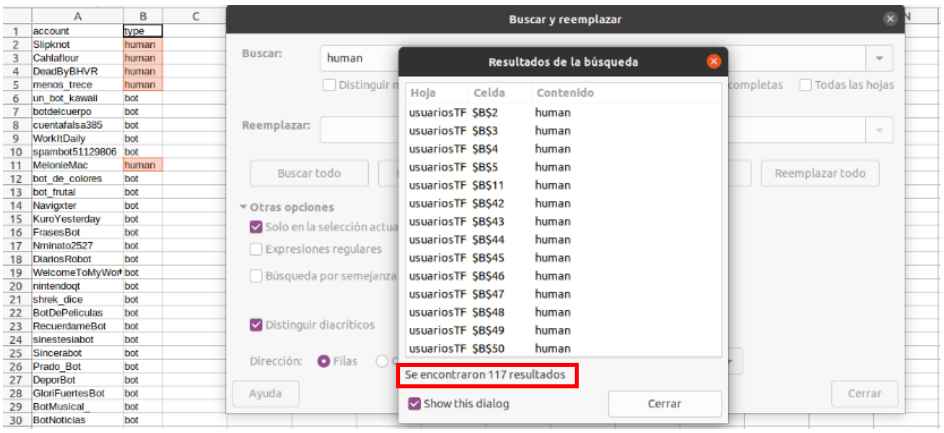


Ilustración 25. Captura con el total de usuarios etiquetados como humanos (Fuente propia)

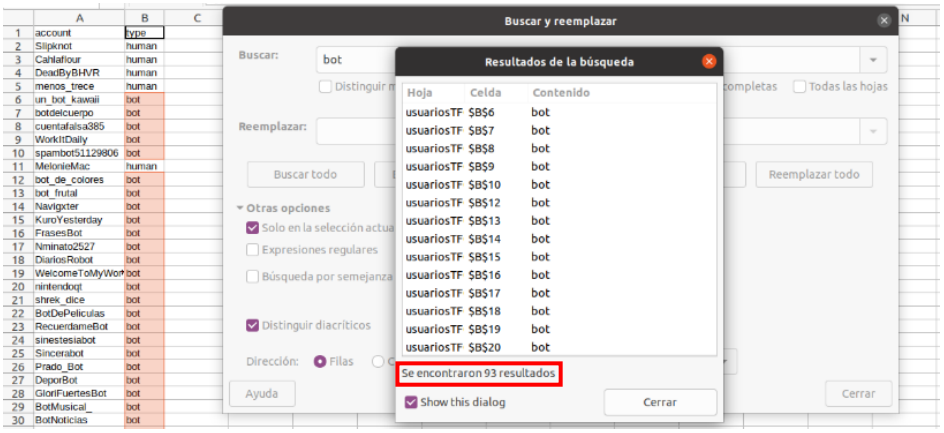


Ilustración 26. Captura con el total de usuarios etiquetados como bots (Fuente propia)

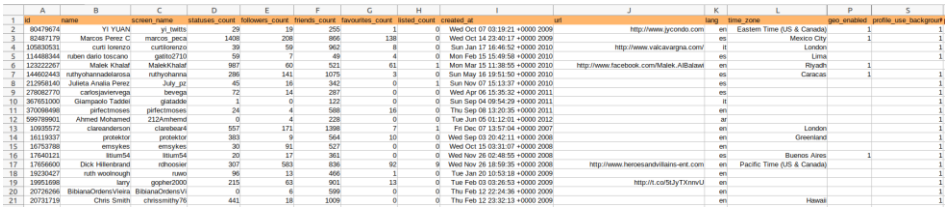
7.2. Otros DataSets

Seguidamente, se buscaron fuentes de datos procedentes de comunidades y/o proyectos científicos que tuvieran listados de usuarios de la red social Twitter previamente etiquetados como bots y humanos.

En este caso, finalmente se seleccionaron datasets recopilados, por un lado, de proyectos MIB y, por otro lado, de la plataforma Kaggle destinada a fomentar el aprendizaje sobre ciencia de datos y machine learning:

- En primer lugar, los datasets procedentes de proyectos MIB se dividían en dos conjuntos principales: el primero constaba de varios ficheros con listados de cuentas reales, seguidores falsos y bots de spam que fueron registradas por colaboradores de

CrowdFlower [30] y el segundo consistía en un conjunto de archivos con listados de seguidores falsos y cuentas reales [31]. Estos datasets listaban miles de cuentas junto con las características presentes en sus perfiles y, además, todos estos ficheros presentaban el mismo formato de columnas. Sin embargo, no presentaban campos de etiquetado, pues **la separación entre bots y humanos se llevaba a cabo organizando los datos de un tipo u otro en ficheros distintos.**



	A	B	C	D	E	F	G	H	I	J	K	L	P	S
1	id	name	screen_name	statuses_count	followers_count	friends_count	favourites_count	listed_count	created_at	url	lang	time_zone	geo_enabled	profile_use_background
2	80479674	YI YUAN	yi_twitts	29	19	255	1	0	Wed Oct 07 03:19:21 +0000 2009	http://www.jycondo.com	en	Eastern Time (US & Canada)	1	1
3	82487179	Marcos Perez C	marcos_peca	1406	208	866	138	0	Wed Oct 14 23:40:17 +0000 2009		es	Mexico City	1	1
4	105830531	curti lorenzo	curtilorenzo	39	59	962	8	0	Sun Jan 17 16:46:52 +0000 2010	http://www.valcavargna.com/	it	London		1
5	114488344	ruben dario toscano	gatito2710	59	7	49	4	0	Mon Feb 15 15:49:58 +0000 2010		es	Lima		1
6	123222267	Malek Khalaf	Malekkhalaf	987	60	521	61	1	Mon Mar 15 11:38:55 +0000 2010	http://www.facebook.com/Malek.AlBalawi	en	Riyadh	1	
7	144602443	ruthyohannadelarosa	ruthyohanna	286	141	1075	3	0	Sun May 16 19:51:50 +0000 2010		es	Caracas	1	1
8	212958140	Julieta Analia Perez	July_jp	45	16	342	0	1	Sun Nov 07 15:13:37 +0000 2010		es			1
9	279082770	carlosjaviervega	bevega	72	14	287	0	0	Wed Apr 06 15:35:32 +0000 2011		es			1
10	367651000	Giampaolo Taddei	gtaddei	1	0	122	0	0	Sun Sep 04 09:54:29 +0000 2011		it			1
11	370098498	pirfectmoses	pirfectmoses	24	4	588	16	0	Thu Sep 08 13:20:35 +0000 2011		en			1
12	599789901	Ahmed Mohamed	212Ahmend	0	4	228	0	0	Tue Jun 05 01:12:01 +0000 2012		ar			1
13	10935572	clareanderson	clarebear4	557	171	1398	7	1	Fri Dec 07 13:57:04 +0000 2007		en	London		1
14	16119337	protektor	protektor	383	9	564	10	0	Wed Sep 03 20:42:11 +0000 2008		en	Greenland		1
15	16753788	emsykes	emsykes	30	91	527	0	0	Wed Oct 15 03:31:07 +0000 2008		en			1
16	17640121	ittum54	ittum54	20	17	361	0	0	Wed Nov 26 02:48:55 +0000 2008		es	Buenos Aires	1	1
17	17656600	Dick Hillenbrand	nthoosier	307	583	836	92	9	Wed Nov 26 18:59:35 +0000 2008	http://www.heroesandvillains-ent.com	en	Pacific Time (US & Canada)		1
18	19230427	ruth woolnough	ruwo	96	13	466	1	0	Tue Jan 20 10:53:18 +0000 2009		en			1
19	19951698	larry	gopher2000	215	63	901	13	0	Tue Feb 03 03:26:53 +0000 2009	http://t.co/5tJyTXonvU	en			1
20	20726266	BibianaOrdensVieira	BibianaOrdensVi	0	6	599	0	0	Thu Feb 12 22:24:36 +0000 2009		en			1
21	20731719	Chris Smith	chrissmithy76	441	18	1009	0	0	Thu Feb 12 23:32:13 +0000 2009		en	Hawaii		1

Ilustración 27. Captura de los 20 primeros registros de uno de los ficheros de MIB (Fuente propia)

- En segundo lugar, de la comunidad Kaggle se recopilaron varios conjuntos de datos, donde:
    - El primero consistía en un fichero compuesto por más de 30000 cuentas de Twitter. Este dataset presentaba dos únicos campos: el identificador del usuario y un campo "account_type" donde **se etiquetaba a cada usuario como bot o humano** [32].

	A	B
1	id	account_type
2	787405734442959000	bot
3	796216118331310000	human
4	875949740503859000	human
5	756119643622736000	human
6	464781334	human
7	800833795234611000	bot
8	55050269	human
9	809500704	human
10	1015021614	human
11	492306486	human
12	432537664	human
13	2579497028	bot
14	1966087(	human
15	313087874	human
16	211550281	human
17	769392715	human
18	772106131875500000	human
19	772521707810787000	human
20	476691783	human
21	384565337	bot

Ilustración 28. Captura de los 20 primeros registros del primer conjunto de Kaggle (Fuente propia)

    - El segundo conjunto resultaba ser un archivo que contenía un listado de más de 2000 cuentas de la red social junto con sus características y un campo "bot" donde los registros pertenecientes **a personas se marcaban con el valor 0 y los de bots con el valor 1** [33].

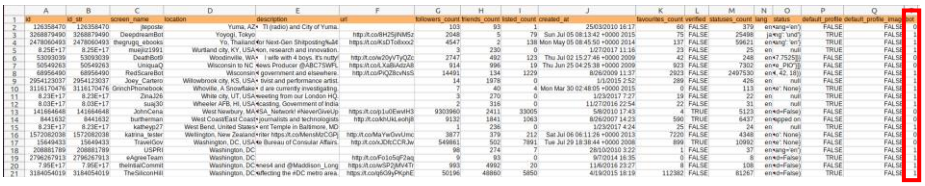


Ilustración 29. Captura de los 20 primeros registros del segundo conjunto de Kaggle (Fuente propia)

- El tercero consistía en un fichero formado por más de 35000 registros pertenecientes a cuentas de Twitter junto con sus características y un campo "account_type" donde **se etiquetaba a cada usuario como bot o humano** [34].

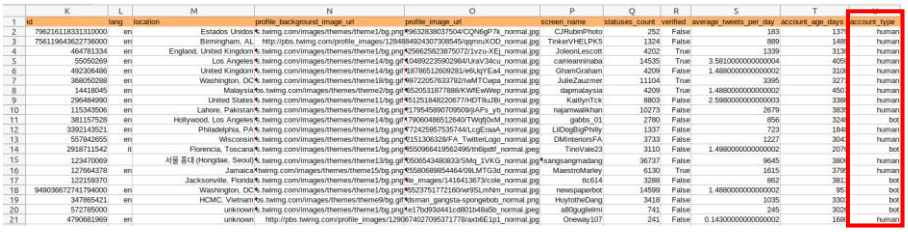


Ilustración 30. Captura de los 20 primeros registros del tercer conjunto de Kaggle (Fuente propia)

- Finalmente, el cuarto conjunto estaba compuesto por dos ficheros donde se recopilaban más de 2000 cuentas junto con sus características y un campo "bot" donde los registros pertenecientes a **personas se marcaban con el valor 0 y los de bots con el valor 1** [35].

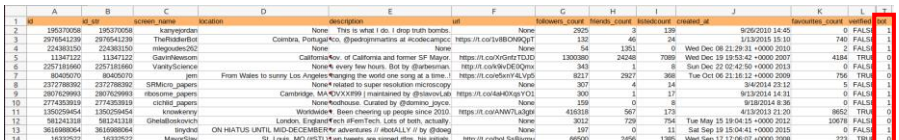


Ilustración 31. Captura de los 20 primeros registros del cuarto conjunto de Kaggle (Fuente propia)

De esta forma, tal y como se puede apreciar en las descripciones de cada conjunto de datos seleccionado, no todos ellos presentaban el mismo número de campos ni tampoco un formato de etiquetado homogéneo, pues unos lo indicaban con un nombre, otros con valores numéricos o incluso organizándolos en ficheros distintos. Asimismo, no todos los campos presentes en los datasets resultaban ser de utilidad, pues en este caso únicamente se necesitaba el identificador de cada cuenta y su correspondiente etiqueta. Por ello, era necesario unificar todos los conjuntos de datos seleccionados en un único fichero con el fin de otorgarle un formato

homogéneo a su etiquetado y, además, prescindir de todos aquellos campos que no fueran necesarios.

7.3. Unificación de usuarios y homogenización del etiquetado

En este apartado se procede a unificar todos los datasets recopilados en uno solo, así como almacenar únicamente las dos columnas que resultan de interés y dar un formato homogéneo en su etiquetado a cada uno de los registros. Así pues, para ello, se decide hacer uso de la herramienta Pentaho Data Integration con el fin de elaborar una transformación que genere el dataset esperado. De esta manera, la transformación finalmente realizada presenta la siguiente estructura:

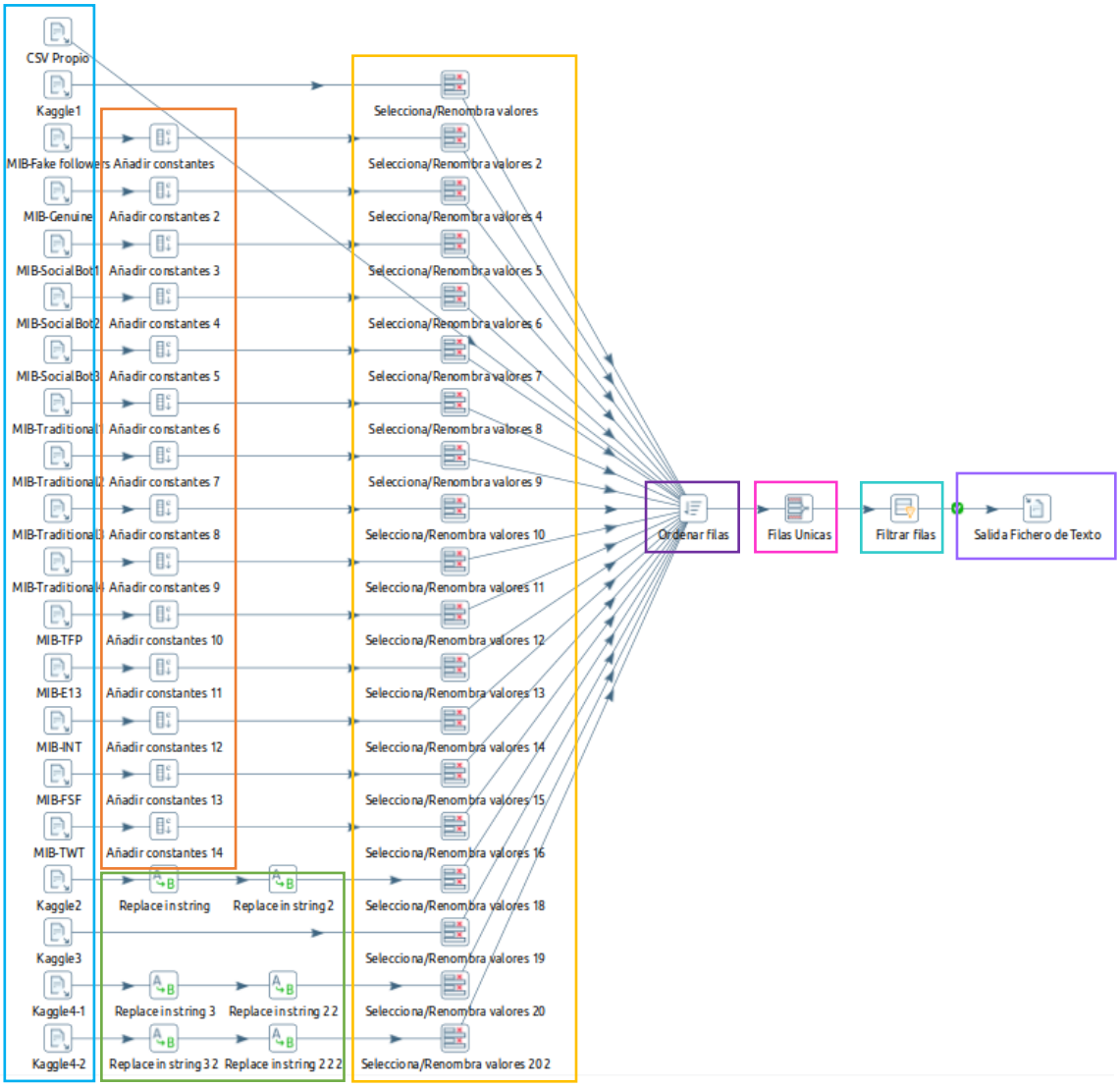


Ilustración 32. Transformación realizada para unir todos los datasets recopilados (Fuente propia)

La transformación consta de varios pasos y, por tanto, se procede a explicar en detalle cada uno de los mismos:

- **CSV file input (■):** Se trata de la entrada donde se especifica la ruta correspondiente a cada fichero CSV que se desea emplear en la transformación. Además, se debe indicar cómo se realiza la separación del contenido dentro del archivo, así como traer las columnas presentes en el mismo y sus tipos de datos.
- **Añadir constantes (■):** Este paso se ha utilizado únicamente con los datasets procedentes de proyectos MIB ya que, como se vio con anterioridad, no poseían ningún tipo de etiquetado como tal y diferenciaban bots o humanos utilizando ficheros distintos. Así pues, mediante este paso se añade una constante "type" cuyo valor sea "bot" o "human" según corresponda en cada caso.
- **Replace in string (■):** Se trata de un paso de transformación empleada sobre aquellos ficheros procedentes de Kaggle donde el etiquetado de bots se indicaba mediante los valores numéricos 1 y 0. Así pues, en él se procede a reemplazar dichos valores por la cadena de texto equivalente, es decir "bot" en lugar de 1 y "human" en vez de 0.
- **Selecciona/renombra valores (■):** En este paso se procede a seleccionar únicamente las dos columnas (identificador y etiqueta) que interesa conservar, con el fin de prescindir del resto de campos que no resultan necesarios.
- **Ordenar filas (■):** Se trata de un paso destinado a ordenar los registros de forma ascendentes según su campo "id", algo necesario para que el siguiente paso "Filas únicas" funcione de forma correcta.
- **Filas únicas (■):** Este paso se encarga de unir todos los datasets tomados como entrada y eliminar aquellos registros que estén duplicados.
- **Filtrar filas (■):** En este paso de la transformación se procede a eliminar aquellas filas que estén vacías.
- **Salida Fichero de Texto (■):** Se trata del último paso de la transformación donde se indican los campos que tendrá el fichero resultante, así como su nombre, ruta y extensión, en este caso "CSV".

De esta forma, una vez elaborada la transformación, se procede ejecutarla y en la pantalla de "Resultados de ejecución" aparecerán el número de registros totales presentes en el fichero de salida y el aviso de que la transformación ha finalizado. En este caso, tras la unificación de los datasets y la eliminación de duplicados, se obtuvieron finalmente un total de 56072 registros de cuentas.

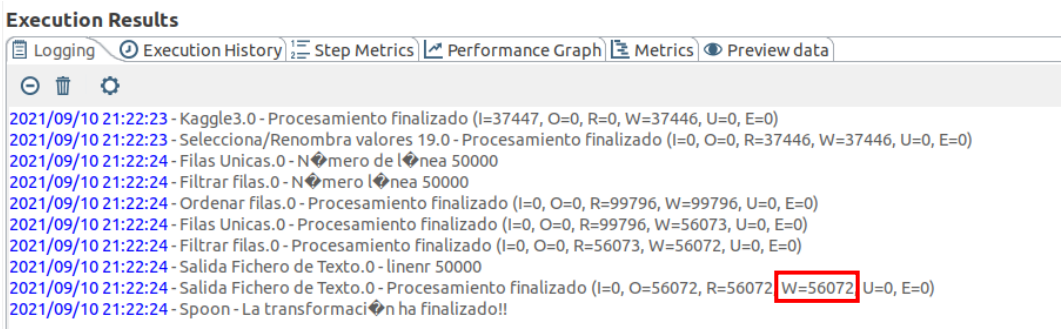


Ilustración 33. Resultados obtenidos tras ejecutar la transformación
(Fuente propia)

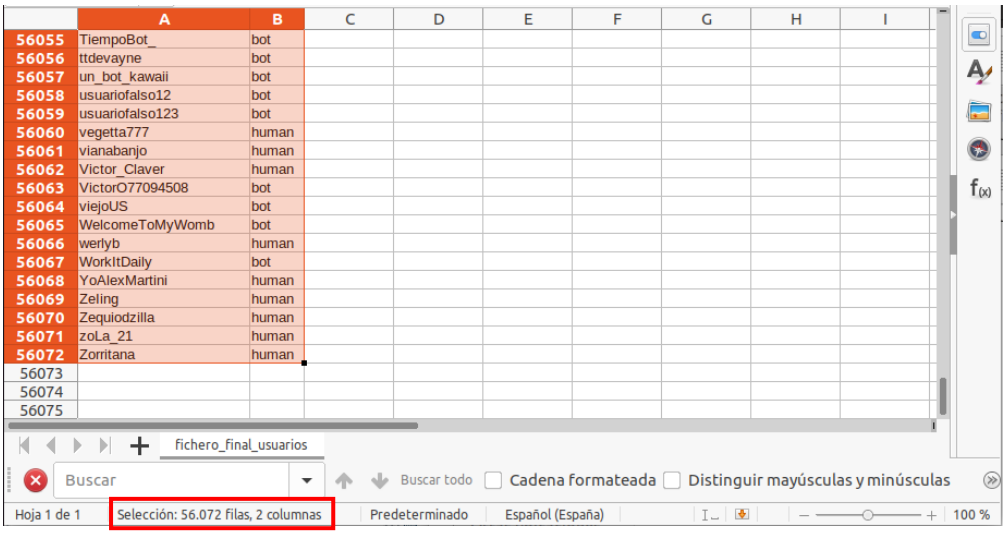


Ilustración 34. Últimos registros del fichero resultante de la transformación
(Fuente propia)

7.4. Extracción de características de usuarios vía API

En este apartado se procede a explicar en detalle cómo se ha llevado a cabo la extracción de características para cada uno de los perfiles mediante la API de Twitter. Así pues, para ello, será necesario disponer de una cuenta de desarrollador autorizada que nos permita acceder al contenedor de datos que ofrece la plataforma. Una vez se disponga de dicha cuenta, se deberá dar de alta una nueva "App" en el sistema con el fin de obtener así los tokens y claves proporcionados por Twitter que permitirán el acceso a la API.

De esta forma, una vez obtenidas las claves, se procede a establecer la conexión mediante la librería Tweepy, creando para ello una instancia de OAuthlander que recibirá como parámetro estas claves de consumo.

```
auth = tweepy.OAuthHandler(consumer_key,consumer_secret_key)
api = tweepy.API(auth, wait_on_rate_limit=True)
```

Código 1. Conexión con la API utilizando Tweepy

Así, una vez se haya hecho efectiva la conexión con la API, deberemos acceder al fichero que contenga el listado de usuarios sobre los cuales queremos extraer la información. Para llevar a cabo esta lectura de datos, se usará la librería "csv" de Python utilizando la función "reader" que recibirá como parámetro el fichero con el listado de usuarios y el símbolo empleado como delimitador de registros.

```
with
open('/home/merm11/PycharmProjects/tfmProject/fichero_final_usuarios.csv') as
origenDatos:
    cuentas = csv.reader(origenDatos, delimiter=';')
```

Código 2. Lectura del fichero con el listado de usuarios

De esta forma, una vez obtenidas las cuentas recopiladas, se procede a recorrer este listado y a ir llamando al método "get_user" de la API con el fin de obtener su objeto completo con toda la información de cada cuenta.

Por otro lado, como en el fichero se disponía también de la etiqueta de cada usuario, se procede a almacenarla en una variable denominada "tipo". Además, se llevará la cuenta tanto del número total de perfiles falsos como de activos que se han procesado.

```
suspendidas = 0 #cuentas que no funcionan (no existen o suspendidas)
activas = 0 #cuentas que si funcionan

#fecha de la medición
hoy_medicion = datetime.now()
fecha_hora_medicion = hoy_medicion.strftime('%Y-%m-%d %H:%M:%S')

for cuenta in cuentas:
        try:
                user = api.get_user(cuenta[0])
        tipo = cuenta[1]
                activas += 1
```

Código 3. Obtención de los objetos usuario

Así, una vez obtenido cada objeto usuario, se extraen las características de su perfil que resulten de interés para el estudio. En este caso se recopilan un total de 18 características.

```
# 1.Id
id = user.id

# 2. Nombre de usuario
nombre = user.name

# 3. Nombre de pantalla del usuario
nombre_pantalla = user.screen_name
```

```
# 4. Localización del usuario
localizacion_usuario = user.location

# 5. URL de la cuenta
url = user.url

# 6. Descripción de la cuenta
descripción = str(user.description).replace("\n", "")

# 7. Indica si la cuenta esta protegida o no
cuenta_protegida = user.protected

# 8. Indica si la cuenta esta verificada o no
cuenta_verificada = user.verified

# 9. Número de seguidores
numero_seguidores = user.followers_count

# 10. Número de usuarios seguidos por la cuenta
numero_usuarios_seguidos = user.friends_count

# 11. Numero de listas públicas de las que el usuario es miembro
numero_listas = user.listed_count

# 12. Número de favoritos dados por el usuario
numero_favoritos = user.favourites_count

# 13. Cantidad de tuits y retuits emitidos por el usuario
numero_tweets_rts = user.statuses_count

# 14. Fecha de creación de la cuenta
fecha_creacion = user.created_at

# 15. URL asociada al banner subido al perfil
try:
    url_banner_perfil = user.profile_banner_url
except AttributeError as error:
    url_banner_perfil = "None"

# 16. URL asociada a la imagen de perfil
url_imagen_perfil = user.profile_image_url_https

# 17. Perfil por defecto
perfil_por_defecto = user.default_profile

# 18. Imagen por defecto de perfil
imagen_por_defecto = user.default_profile_image
```

Código 4. Extracción de las características de perfil de los usuarios

Seguidamente, una vez extraídas y almacenadas en variables las características de interés de cada cuenta, se procede a almacenarlas en un fichero. Así, el contenido de este constará de los siguientes contenidos: los identificadores propios de cada usuario, sus características, así como su tipo (bot o humano).

Así pues, para llevar a cabo esta escritura de datos, se usará nuevamente la librería “csv” con la función “writer” que recibirá como parámetros el fichero con el listado de usuarios y el símbolo empleado como delimitador de registros.

```
# fichero destino usuarios encontrados
ruta_fichero_salida='/home/merm11/PycharmProjects/tfmProject/fichero_caracter
isticas.csv'

with open(ruta_fichero_salida,'a',newline='') as f_datos:
        writer_datos = csv.writer(f_datos,delimiter=';',quotechar='"')

        # Si el fichero está vacío se insertan nombres de las columnas
        if os.stat(ruta_fichero_salida).st_size == 0:
        writer_datos.writerow([“param1”, “param2”,… “paramN”])

        # Escritura de los datos
        writer_datos.writerow([variable1, variable2, … , variableN])
```

Código 5. Escritura de las características extraídas en un fichero

Es posible que varias de las cuentas presentes en el listado, especialmente en el caso de aquellos que sean de tipo “bot”, ya no existan o hayan sido suspendidas de la plataforma Twitter. Por ello, se procede a capturar la excepción y, para todas aquellas cuentas en las que falle la petición “get_user” se escribirá en un fichero el identificador de la cuenta, el tipo (bot o humano) y el mensaje de error recibido, pudiendo de esta forma conocer qué y cuantos usuarios han quedado descartados de la extracción.

Así pues, para llevar a cabo esta escritura de datos, se usará nuevamente la librería “csv” con la función “writer” que recibirá como parámetros el fichero con el listado de usuarios y el símbolo empleado como delimitador de registros.

```
except Exception as e:

        suspendidas += 1
        # fichero destino usuarios fallidos (no existen o suspendidos)
        ruta_usuarios_fallidos =
        '/home/merm11/PycharmProjects/tfmProject/fichero_errores.csv'

        with open(ruta_usuarios_fallidos,'a',newline='') as f_fallidos:
                writer_fallidos =
                csv.writer(f_fallidos,delimiter=';',quotechar='"')

        if os.stat(ruta_usuarios_fallidos).st_size == 0:
                writer_fallidos.writerow(["cuenta", "tipo",
                "Excepcion"])

        writer_fallidos.writerow([cuenta[0], tipo, str(e)])
```

Código 6. Control de excepciones para cuentas inexistentes o suspensas

Para cuando el proceso de extracción finalice, se ha preparado un conjunto de mensajes por pantalla donde visualizar información relacionada con la ejecución del proceso como la fecha de inicio y fin de la ejecución, el tiempo total transcurrido en minutos, la cantidad total de cuentas leídas, de cuentas activas y de cuentas suspendidas.

```
# fecha y hora de fin del proceso
hoy_fin = datetime.now()
fecha_hora_fin = hoy_fin.strftime('%Y-%m-%d %H:%M:%S')

#tiempo total transcurrido
tiempo_transcurrido = hoy_fin - hoy_inicio
minutos = divmod(tiempo_transcurrido.total_seconds(), 60)

print("------Proceso finalizado-------------------------------------")
print("     * Fecha y hora de inicio del proceso: ", fecha_hora_inicio)
print("     * Fecha y hora de finalización del proceso: ", fecha_hora_fin)
print("     * Tiempo total transcurrido: ", minutos[0], 'minutos',
minutos[1], 'segundos')
print("     * Total cuentas: ", activas+suspendidas)
print("     * Total cuentas activas: ",activas)
print("     * Total cuentas suspendidas: ",suspendidas)
```

Código 7. Conjunto de mensajes a mostrar al finalizar la extracción

Así pues, una vez finalizada la ejecución del proceso cuya duración aproximada fue de 2 días, se obtuvo la siguiente salida por pantalla.

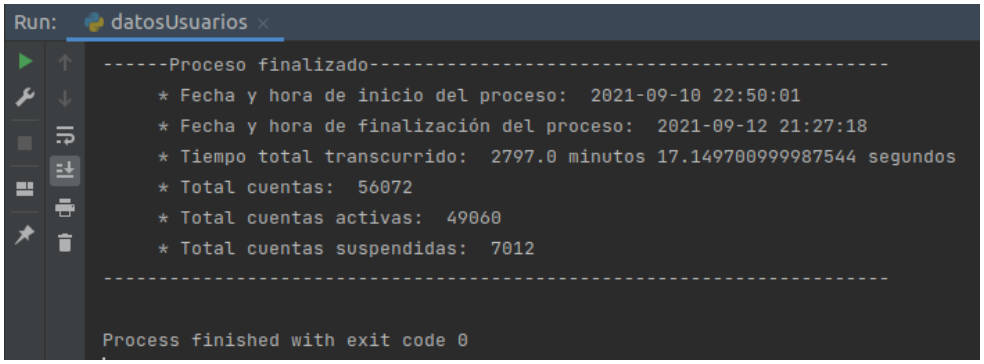


Ilustración 35. Mensaje tras finalizar la ejecución del proceso de extracción (Fuente propia)

De esta forma, tal y como se puede observar en la anterior captura, el número de cuentas totales coincide de forma correcta con el total de cuentas unificadas con anterioridad mediante la herramienta Pentaho, haciendo un total de 56072 cuentas leídas de forma correcta.

Seguidamente, se procede a comprobar los ficheros resultantes tras el proceso, comenzando con el fichero de las características extraídas de las cuentas activas. De esta forma, si se consulta el archivo se puede observar como el recuento de 49060 registros coincide con el mostrado con anterioridad por la terminal y, por tanto, se verifica que la escritura se ha efectuado de forma correcta.

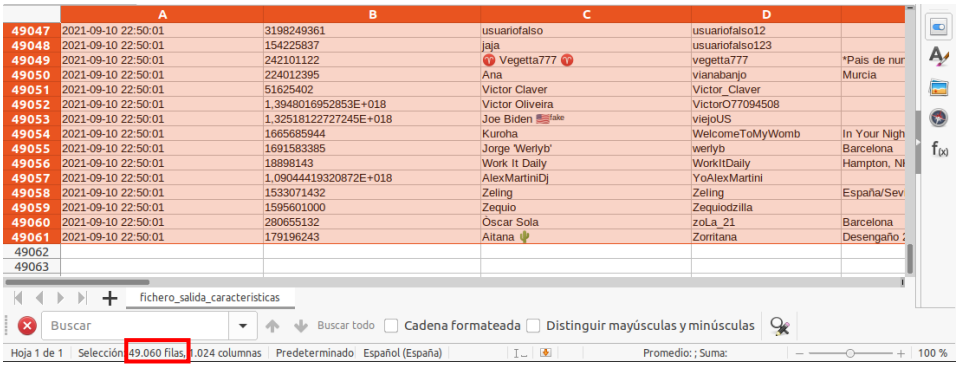


Ilustración 36. Cantidad total de filas en el fichero de características resultante de la extracción (Fuente propia)

Asimismo, se realiza una búsqueda por los nombres “bot” y “human” sobre la columna “tipo” con el fin de obtener la cantidad total de muestras disponibles de cada tipo, tal y como se muestra en las siguientes capturas.

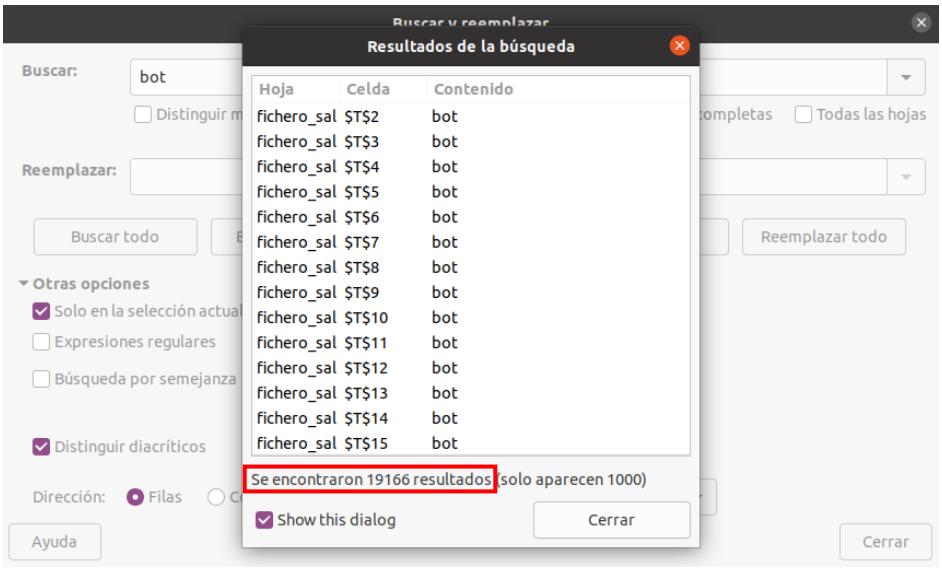


Ilustración 37. Captura con la cantidad de registros que presentan como tipo "bot" (Fuente propia)

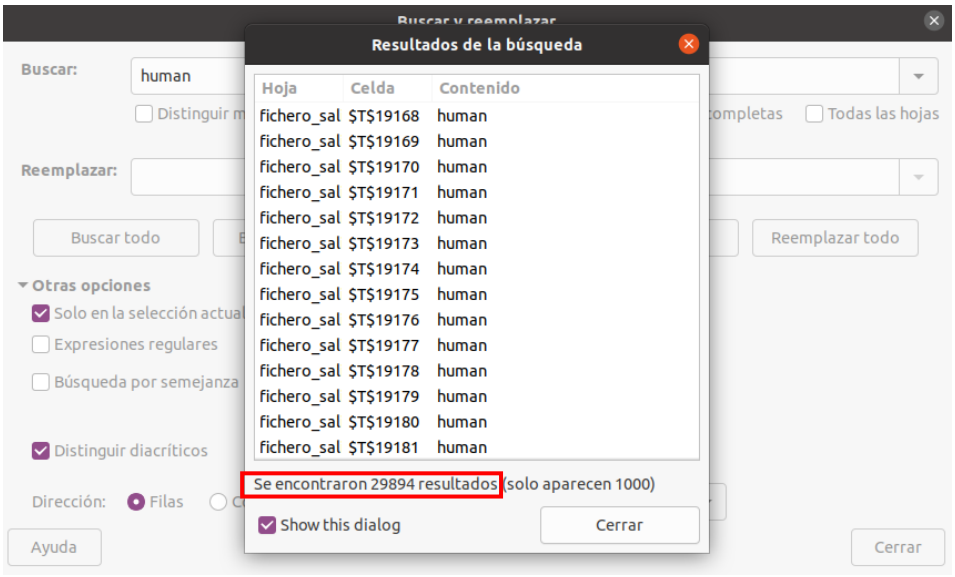


Ilustración 38.Captura con la cantidad de registros que presentan como tipo "human" (Fuente propia)

De esta manera, de las 49060 cuentas recopiladas de manera correcta a través de la API de Twitter, 19166 se corresponden con bots y las restantes 29894 con humanos.

Tras ello, se comprueba el fichero que almacena las cuentas suspensas o usuarios fallidos junto con los errores, donde nuevamente se puede apreciar que la cantidad de 7012 registros coincide con aquella anteriormente mostrada por la terminal, verificando por tanto que esta escritura también es correcta.

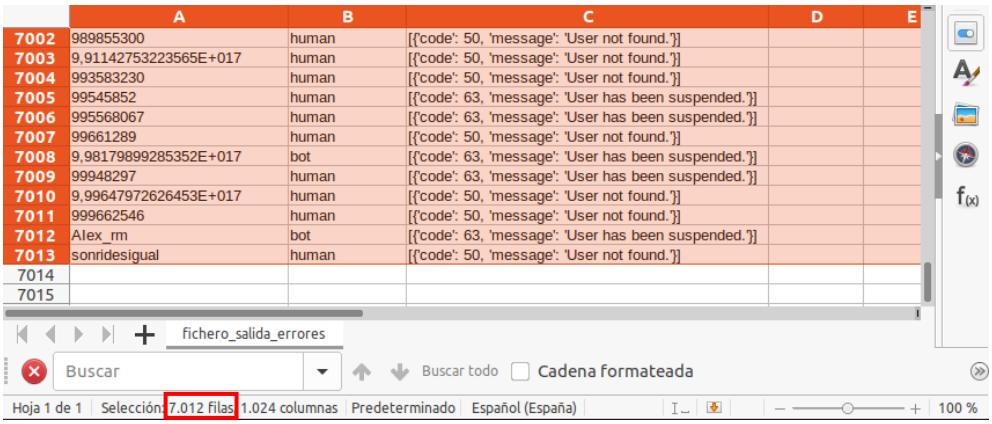


Ilustración 39. Cantidad total de filas en el fichero resultante de los usuarios fallidos de la extracción (Fuente propia)

8. Métodos de selección de características

En este apartado se procede a hablar en detalle sobre qué son los métodos de selección de características y cómo se aplican algunos de ellos en este proyecto. Estos métodos tienen como objetivo la reducción de los variables de entrada para los modelos de predicción, con el fin de conservar únicamente aquellos atributos que sean más relevantes para predecir la variable objetivo [36]. En el caso de este proyecto, las variables de entrada se corresponden con las características recopiladas de los distintos usuarios y la variable objetivo resulta ser la columna "tipo" que indica si se trata de un bot o un humano.

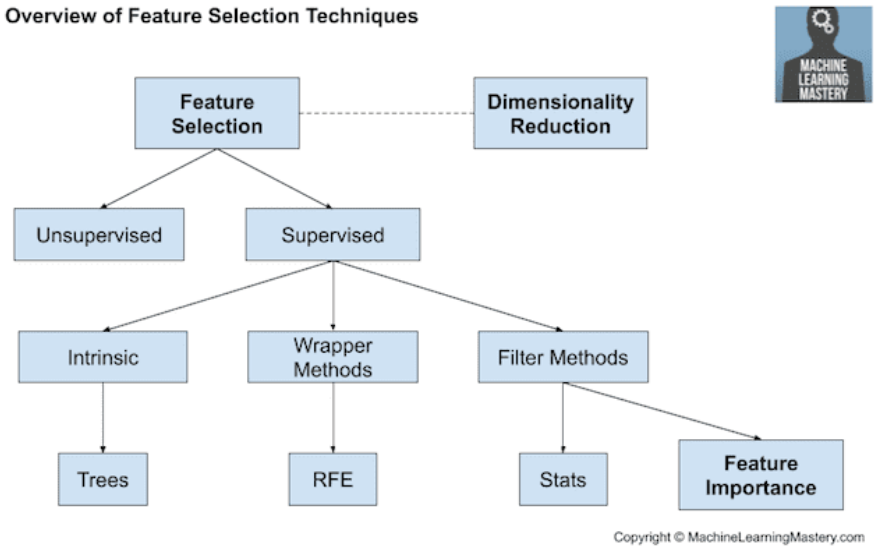


Ilustración 40.Jerarquía de técnicas de selección de características (Fuente https://machinelearningmastery.com/)

De esta manera, se distinguen dos tipos principales de selección de características en función de si tienen en cuenta o no la variable objetivo:

- **Métodos no supervisados:** no tienen en cuenta la etiqueta o variable objetivo.
- **Métodos supervisados:** si hacen uso de la variable objetivo.

En este caso, nos centraremos en los métodos supervisados puesto que en este proyecto si importa la variable objetivo, en este caso la columna "tipo" de nuestro dataset. Así, dentro de las técnicas de selección de características supervisadas se distinguen 3 principales:

- **Intrínsecas,** mediante las cuales se realiza la selección de características durante el aprendizaje. Un ejemplo de ello sería arboles de decisión.

- **Métodos de contenedor o envoltura,** los cuales generan diversos subconjuntos de variables de entrada, escogiendo y priorizando aquellos mejoren el rendimiento. Un ejemplo de ello sería RFE.
- **Métodos de filtro,** mediante los cuales se seleccionan subconjuntos de variables entrada en función del vínculo que presenten con la variable objetivo. Así, para valorar esta relación entre variables, utilizan técnicas estadísticas que les permiten puntuar y seleccionar qué características utilizar. Un par de ejemplos de ello serían los métodos de estadística y los métodos de importancia de características.

Así, antes de escoger cualquier técnica, es necesario analizar si el problema a abordar resulta ser una tarea de clasificación o de regresión, pues ello dependerá del tipo de resultado que se busque en cada caso [37]:

- Por un lado, se entienden como **tareas de clasificación** aquellas que tienen como resultado un número limitado de más o menos categorías según la problemática. De esta forma, algunos ejemplos de este tipo de problemas podrían ser los siguientes:
    - ¿Tipo de bicicleta? [Carretera, Montaña]
    - ¿Color de pelo? [Rubio, moreno, pelirrojo, castaño]
- Por otro lado, se entienden como **tareas de regresión** aquellas que tienen como resultado un número de infinitos resultados. De esta forma, algunos ejemplos de este tipo de problemas podrían ser los siguientes:
    - Tiempo estimado de llegada de un autobús a su parada.
    - Cantidad estimada de entradas que se esperan vender para un espectáculo.

En este caso, el presente proyecto actual busca llevar a cabo la predicción sobre si una cuenta resulta ser bot o humano en la red social Twitter. Por tanto, puesto que se trata de un problema con un número acotado de resultados, **estamos ante una tarea de clasificación.**

De esta forma, se propone hacer uso de dos técnicas de selección de características para la experimentación de este trabajo: una basada en métodos de filtro, medidas estadísticas, y otra basada en técnicas de contenedor, RFE.

8.1. Medidas estadísticas

El uso de técnicas estadísticas es bastante frecuente y depende fuertemente de los tipos de datos que presenten tanto las variables de entrada como de salida. Así, se distinguen

principalmente dos tipos: variables categóricas (nominales, ordinales y booleanos) y variables numéricas (enteros y flotantes).

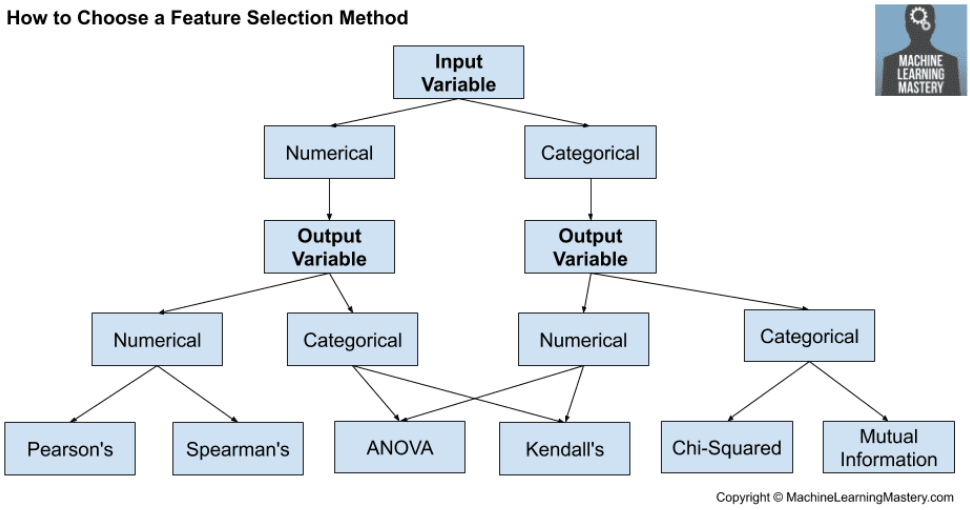


Ilustración 41. Métodos de selección según los tipos de variables (Fuente https://machinelearningmastery.com/)

En este caso se pretende hacer uso de una de las técnicas más comunes para la selección de características relevantes en un problema de clasificación, el coeficiente de correlación ANOVA, el cual requiere de variables numéricas de entrada y categóricas de salida. De esta forma, para poder aplicarlo, ya que el dataset generado para este proyecto presenta variables de entrada tanto categóricas como numéricas, será necesario transformarlas en su totalidad a variables numéricas con el fin de poder hacer uso de la técnica mencionada.

Así pues, en primer lugar, se procede a cargar el fichero de datos, seleccionando únicamente las columnas referentes a características y etiquetado, ya que la fecha de medición y el identificador de cada usuario no resultan de utilidad para este procedimiento.

```
import pandas as pd
#Carga de datos
path = '/home/merm11/Escritorio/fichero_salida_caracteristicas.csv'
delimiter = ';'

df = pd.read_csv(path,delimiter,
                 usecols=['nombre','nombre_pantalla','localizacion_usuario',
                       'url','descripcion', 'cuenta_protegida',
                        'cuenta_verificada','numero_seguidores',
                        'numero_usuarios_seguidos','numero_listas',
                        'numero_favoritos','numero_tweets_rts',
                        'fecha_creacion','url_banner_perfil',
                        'url_imagen_perfil', 'perfil_por_defecto',
                        'imagen_por_defecto','tipo'
                       ])
```

Código 8. Carga de datos del fichero que contiene las características

Una vez cargadas las características, se procede a llamar al método “info” de la clase “DataFrame” con el fin de visualizar el tipo de dato que presenta cada una de las variables. Así, tal y como se puede observar en la siguiente captura, existen datos de tipo ‘object’ y ‘bool’ entre las variables de entrada que habrá que convertir a valores numéricos mediante codificación. En el caso de la variable de salida “tipo” no requerirá de codificación, puesto que la técnica del coeficiente de correlación ANOVA espera una variable de salida de tipo categórico.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49060 entries, 0 to 49059
Data columns (total 18 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   nombre                    49049 non-null  object
 1   nombre_pantalla           49060 non-null  object
 2   localizacion_usuario      30752 non-null  object
 3   url                       21606 non-null  object
 4   descripcion               39783 non-null  object
 5   cuenta_protegida          49060 non-null  bool
 6   cuenta_verificada         49060 non-null  bool
 7   numero_seguidores         49060 non-null  int64
 8   numero_usuarios_seguidos  49060 non-null  int64
 9   numero_listas             49060 non-null  int64
 10  numero_favoritos          49060 non-null  int64
 11  numero_tweets_rts         49060 non-null  int64
 12  fecha_creacion            49060 non-null  object
 13  url_banner_perfil         49060 non-null  object
 14  url_imagen_perfil         49053 non-null  object
 15  perfil_por_defecto        49060 non-null  bool
 16  imagen_por_defecto        49060 non-null  bool
 17  tipo                      49060 non-null  object
dtypes: bool(4), int64(5), object(9)
memory usage: 5.4+ MB
```

Ilustración 42. Variables y tipos de datos del dataset
(Fuente propia)

De esta forma, para llevar a cabo la codificación de las variables de entrada, se hará uso del codificador LabelEncoder únicamente sobre aquellas variables de entrada con tipo categórico, tal y como puede observarse en el siguiente fragmento de código. Este método nos lo facilitará la librería “sklearn” destinada a tareas de Machine learning que ya fue instalada con anterioridad.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df['nombre'] = encoder.fit_transform(df.nombre.values)
df['nombre_pantalla'] = encoder.fit_transform(df.nombre_pantalla.values)
df['localizacion_usuario']=encoder.fit_transform(df.localizacion_usuario.valu
es)
df['url'] = encoder.fit_transform(df.url.values)
df['descripcion'] = encoder.fit_transform(df.descripcion.values)
df['cuenta_protegida'] = encoder.fit_transform(df.cuenta_protegida.values)
df['cuenta_verificada'] = encoder.fit_transform(df.cuenta_verificada.values)
df['fecha_creacion'] = encoder.fit_transform(df.fecha_creacion.values)
df['url_banner_perfil'] = encoder.fit_transform(df.url_banner_perfil.values)
```

```
df['url_imagen_perfil'] = encoder.fit_transform(df.url_imagen_perfil.values)
df['perfil_por_defecto'] =
encoder.fit_transform(df.perfil_por_defecto.values)
df['imagen_por_defecto'] =
encoder.fit_transform(df.imagen_por_defecto.values)
```

Código 9. Codificación de variables para poder aplicar los métodos de selección

Una vez finalizada la codificación, si se consulta el método "info" de la clase "DataFrame", los tipos de datos de las variables de entrada ya deberán aparecer como tipo numérico.

```
Variables codificadas a tipo numerico:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49060 entries, 0 to 49059
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   nombre                   49060 non-null  int64
 1   nombre_pantalla          49060 non-null  int64
 2   localizacion_usuario     49060 non-null  int64
 3   url                      49060 non-null  int64
 4   descripcion              49060 non-null  int64
 5   cuenta_protegida         49060 non-null  int64
 6   cuenta_verificada        49060 non-null  int64
 7   numero_seguidores        49060 non-null  int64
 8   numero_usuarios_seguidos 49060 non-null  int64
 9   numero_listas            49060 non-null  int64
 10  numero_favoritos         49060 non-null  int64
 11  numero_tweets_rts        49060 non-null  int64
 12  fecha_creacion           49060 non-null  int64
 13  url_banner_perfil        49060 non-null  int64
 14  url_imagen_perfil        49060 non-null  int64
 15  perfil_por_defecto       49060 non-null  int64
 16  imagen_por_defecto       49060 non-null  int64
 17  tipo                     49060 non-null  object
dtypes: int64(17), object(1)
memory usage: 6.7+ MB
```

Ilustración 43. Variables y tipos de datos del dataset tras codificar
(Fuente propia)

Una vez codificadas todas las variables de entrada a tipo numérico, será necesario dividir las características en variables de entrada (x) y de salida (y).

Tras ello, necesitaremos de nuevo la librería "sklearn" destinada a tareas de Machine learning que ya fue instalada con anterioridad. Esta será utilizada para definir y aplicar una selección de las K mejores variables sobre el dataset utilizando para ello el método "SelectKBest", al cual será preciso indicarle en el parámetro "score_func" que se desea hacer uso del coeficiente de correlación ANOVA "f_classif", así como indicarle en el parámetro "k" que extraiga las mejores 10 características de las 17 existentes.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from numpy import array

#separamos las variables de entrada y salida
```

```
x = df.iloc[:,0:17]
y = df.iloc[:,17].values

#selección de 10 características
seleccionadas = SelectKBest(score_func=f_classif, k=10)
```

Código 10. Método SelectKBest con ANOVA para la selección de las 10 mejores características

Seguidamente, se procede a seleccionar las características y a transformar el conjunto de datos mediante el método "fit_transform". Además, para poder acceder al nombre de las características finalmente seleccionadas, será necesario obtener los índices de las mismas mediante el método "get_support".

```
#se aplica la selección
X_reducida = seleccionadas.fit_transform(x, y)

top10 = seleccionadas.get_support()
caracteristicas = array(x.columns)

print("Todas las características:")
print(caracteristicas)
print("-----------------------------------")
print("Las mejores 10 caracteristicas:")
print(caracteristicas[top10])
```

Código 11. Selección y transformación de características con ANOVA

De esta forma, si se visualiza el resultado por pantalla, se puede observar que las 10 características seleccionadas por el método resultan ser las siguientes: localizacion_usuario, url, descripcion, cuenta_verificada, numero_seguidores, numero_favoritos, fecha_creacion, url_banner_perfil, url_imagen_perfil y perfil_por_defecto.

```
Todas las características:
['nombre' 'nombre_pantalla' 'localizacion_usuario' 'url' 'descripcion'
 'cuenta_protegida' 'cuenta_verificada' 'numero_seguidores'
 'numero_usuarios_seguidos' 'numero_listas' 'numero_favoritos'
 'numero_tweets_rts' 'fecha_creacion' 'url_banner_perfil'
 'url_imagen_perfil' 'perfil_por_defecto' 'imagen_por_defecto']
-----------------------------------
Las mejores 10 caracteristicas:
['localizacion_usuario' 'url' 'descripcion' 'cuenta_verificada'
 'numero_seguidores' 'numero_favoritos' 'fecha_creacion'
 'url_banner_perfil' 'url_imagen_perfil' 'perfil_por_defecto']
```

Ilustración 44. Las mejores 10 variables seleccionadas mediante f_regression (Fuente propia)

8.2. RFE

Se trata de un método de contenedor muy popular cuya finalidad es la selección de características relevantes para la predicción de variables objetivo. Esta técnica puntúa cada una de las características de entrada mediante un algoritmo de aprendizaje automático o un método estadístico y puede ser usado tanto para problemas de regresión como de clasificación. De esta forma, al método se le debe facilitar como parámetros el algoritmo a utilizar y la cantidad de mejores características que deseamos extraer. Además, también es posible realizar esta selección del número de características de forma automática mediante el uso de la clase RFECV, la cual realiza una evaluación de validación cruzada (CV). Así, selecciona de forma automática la cantidad de variables de entrada necesarias para obtener buenos resultados [3838].

De esta forma, puesto que el problema a abordar en este proyecto resulta ser una tarea de clasificación, se procede a aplicar las clases RFE y RFECV utilizando los algoritmos de clasificación de árboles de decisión y de regresión logística. Estas clases y funciones nos las facilitará la librería "sklearn" destinada a tareas de Machine learning que ya fue instalada con anterioridad.

Así pues, en primer lugar, se procede a cargar el fichero de datos, seleccionando únicamente las columnas referentes a características y etiquetado, ya que la fecha de medición y el identificador de cada usuario no resultan de utilidad tampoco para este procedimiento, tal y como puede observarse en "*Código 8. Carga de datos del fichero que contiene las características".*

Una vez cargadas las características, se procede a llamar al método "info" de la clase "DataFrame" con el fin de visualizar el tipo de dato que presenta cada una de las variables. Así, al igual que se observó en el caso anterior en la *"Ilustración 42. Variables y tipos de datos del dataset",* existen datos de tipo 'object' y 'bool' entre las variables de entrada que habrá que convertir a valores numéricos mediante codificación.

Así, para la variable de salida "tipo" no se requerirá de codificación tampoco para esta otra técnica. De esta forma, para llevar a cabo la codificación de las variables de entrada, se hará uso nuevamente del codificador LabelEncoder únicamente sobre aquellas variables de entrada con tipo categórico. Este método nos lo facilitará nuevamente la librería "sklearn" destinada a tareas de Machine learning que ya fue instalada con anterioridad, tal y como puede observarse en "*Código 9. Codificación de variables para poder aplicar los métodos de selección*".

Una vez finalizada la codificación, si se consulta el método "info" de la clase "DataFrame", los tipos de datos de las variables de entrada ya deberán aparecer como tipo numérico, tal y como se vio anteriormente en la *"Ilustración 43. Variables y tipos de datos del dataset tras codificar".*

8.2.1. Selección no automática de características con RFE

En este apartado se procede a hablar en detalle sobre los algoritmos seleccionados para llevar a cabo la elección de características no automática.

8.2.1.1. Regresión logística

Una vez cargadas las características de los usuarios, se procede a dividir las características en variables de entrada (x) y de salida (y). Tras ello, se define y aplica una selección de las 10 mejores variables de las 17 existentes mediante el método "RFE" utilizando para ello la técnica de regresión logística mediante el método "fit". Además, para poder acceder al nombre de las características finalmente seleccionadas, será necesario obtener los índices de las mismas mediante el método "get_support".

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

x = df.iloc[:,0:17]
y = df.iloc[:,17].values

#se define la técnica y se encaja
metodo = LogisticRegression()
rfe = RFE(metodo, 10)
rfe = rfe.fit(x,y)

top10rfe = rfe.get_support()
caracteristicasRFE = array(x.columns)
print("Las mejores 10 caracteristicas [RFE-RegresionLogistica]:")
print(caracteristicasRFE[top10rfe])
```

Código 12. Método RFE para la extracción de las 10 mejores características con regresión logística

De esta forma, si se visualiza el resultado por pantalla, se puede observar que las 10 características seleccionadas por el método resultan ser las siguientes: nombre, nombre_pantalla, localizacion_usuario, url, descripcion, numero_listas, numero_favoritos, fecha_creacion, url_banner_perfil y url_imagen_perfil.

```
Las mejores 10 caracteristicas [RFE-RegresionLogistica]:
['nombre' 'nombre_pantalla' 'localizacion_usuario' 'url' 'descripcion'
 'numero_listas' 'numero_favoritos' 'fecha_creacion' 'url_banner_perfil'
 'url_imagen_perfil']
```

Ilustración 45. Las mejores 10 variables seleccionadas mediante RFE con regresión logística (Fuente propia)

8.2.1.2. Árbol de decisión

Una vez cargadas las características de los usuarios, se procede a dividir las características en variables de entrada (x) y de salida (y). Tras ello, se define y aplica una selección de las 10

mejores variables de las 17 existentes mediante el método “RFE” utilizando para ello la técnica de árbol de decisión mediante el método “fit”. Además, para poder acceder al nombre de las características finalmente seleccionadas, será necesario obtener los índices de las mismas mediante el método “get_support”.

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier

x = df.iloc[:,0:17]
y = df.iloc[:,17].values

#se define la técnica y se encaja
metodo2 = DecisionTreeClassifier()
rfe2 = RFE(metodo2, 10)
rfe2 = rfe2.fit(x,y)

top10rfe2 = rfe2.get_support()
caracteristicasRFE2 = array(x.columns)
print("Las mejores 10 caracteristicas [RFE-ArbolDeDecision]:")
print(caracteristicasRFE2[top10rfe2])
```

Código 13. Método RFE para la extracción de las 10 mejores características con árbol de decisión

De esta forma, si se visualiza el resultado por pantalla, se puede observar que las 10 características seleccionadas por el método resultan ser las siguientes: nombre, nombre_pantalla, localizacion_usuario, descripcion, numero_seguidores, numero_usuarios_seguidos, numero_favoritos, numero_tweets_rts, fecha_creacion y url_imagen_perfil.

```
Las mejores 10 caracteristicas [RFE-ArbolDeDecision]:
['nombre' 'nombre_pantalla' 'localizacion_usuario' 'descripcion'
 'numero_seguidores' 'numero_usuarios_seguidos' 'numero_favoritos'
 'numero_tweets_rts' 'fecha_creacion' 'url_imagen_perfil']
```

Ilustración 46. Las mejores 10 variables seleccionadas mediante RFE con árbol de decisión (Fuente propia)

8.2.2. Selección automática de características con RFECV

En este apartado se procede a hablar en detalle sobre los algoritmos seleccionados para llevar a cabo la elección de características automática.

8.2.2.1. Regresión logística

Una vez cargadas las características de los usuarios, se procede a dividir las características en variables de entrada (x) y de salida (y). Tras ello, la clase RFECV calculará de forma automática las K mejores características de las 17 existentes utilizando para ello la técnica de regresión

logística mediante el método “fit”. Además, para poder acceder al nombre de las características finalmente seleccionadas, será necesario obtener los índices de las mismas mediante el método “get_support”.

```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression

x = df.iloc[:,0:17]
y = df.iloc[:,17].values

metodo_rfecv = LogisticRegression()
rfecv = RFECV(estimator=metodo_rfecv, step=1, cv=5, scoring='accuracy')
rfecv = rfecv.fit(x, y)

print('Número óptimo de características [RFECV-RegresionLogistica]:',
rfecv.n_features_)
print('Mejores características [RFECV-RegresionLogistica]:',
x.columns[rfecv.support_])
```

Código 14. Método RFECV con regresión logística

De esta forma, si se visualiza el resultado por pantalla, se puede observar que el método no estima ningún número óptimo de características a seleccionar, siendo las variables devueltas las 17 existentes.

```
Número óptimo de características [RFECV-RegresionLogistica]: 17
Mejores características [RFECV-RegresionLogistica]: Index(['nombre', 'nombre_pantalla', 'localizacion_usuario', 'url',
       'descripcion', 'cuenta_protegida', 'cuenta_verificada',
       'numero_seguidores', 'numero_usuarios_seguidos', 'numero_listas',
       'numero_favoritos', 'numero_tweets_rts', 'fecha_creacion',
       'url_banner_perfil', 'url_imagen_perfil', 'perfil_por_defecto',
       'imagen_por_defecto'],
      dtype='object')
```

Ilustración 47. Las mejores variables seleccionadas mediante RFECV con regresión logística (Fuente propia)

8.2.2.2. Árbol de decisión

Una vez cargadas las características de los usuarios, se procede a dividir las características en variables de entrada (x) y de salida (y). Tras ello, la clase RFECV calculará de forma automática las K mejores características de las 17 existentes utilizando para ello la técnica de árbol de decisión mediante el método “fit”. Además, para poder acceder al nombre de las características finalmente seleccionadas, será necesario obtener los índices de las mismas mediante el método “get_support”.

```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import DecisionTreeClassifier

x = df.iloc[:,0:17]
y = df.iloc[:,17].values

metodo_rfecv2 = DecisionTreeClassifier()

rfecv2 = RFECV(estimator=metodo_rfecv2, step=1, cv=5, scoring='accuracy')
```

```
rfecv2 = rfecv2.fit(x, y)
print('Número óptimo de características [RFECV-ArbolDeDecision]:',
rfecv2.n_features_)
print('Mejores características [RFECV-ArbolDeDecision]:',
x.columns[rfecv2.support_])
```

Código 15. Método RFECV con árbol de decisión

De esta forma, si se visualiza el resultado por pantalla, se puede observar que el método estima como número óptimo de características a seleccionar 7, siendo las variables seleccionadas las siguientes: descripción, numero_seguidores, numero_usuarios_seguidos, numero_favoritos, numero_tweets_rts y fecha_creacion.

```
Número óptimo de características [RFECV-ArbolDeDecision]: 7
Mejores características [RFECV-ArbolDeDecision]: Index(['descripcion', 'numero_seguidores', 'numero_usuarios_seguidos',
       'numero_favoritos', 'numero_tweets_rts', 'fecha_creacion',
       'url_imagen_perfil'],
      dtype='object')
```

Ilustración 48. Las mejores variables seleccionadas mediante RFECV con árbol de decisión (Fuente propia)

8.3. Comparativa de los resultados obtenidos

En este apartado se procede a comparar las distintas selecciones de características que han realizado las técnicas aplicadas.

Tabla 6. Reducciones de características obtenidas tras aplicar las distintas técnicas

Técnica de selección	Selección automática	Cantidad total de características	Cantidad de características seleccionadas
SelectKBest con el coeficiente de correlación ANOVA	No	17	10
RFE con regresión logística	No	17	10
RFE con árbol de decisión	No	17	10
RFECV con regresión logística	Sí	17	17
RFECV con árbol de decisión	Sí	17	7

Así, tal y como se puede observar en la tabla anterior, todos los métodos aplicados, a excepción de RFECV con regresión logística, han logrado obtener una selección reducida de características óptimas.

Tabla 7. Características seleccionadas para las distintas técnicas aplicadas

Características	SelectKBest con el coeficiente de correlación ANOVA	RFE con regresión logística	RFE con árbol de decisión	RFECV con regresión logística	RFECV con árbol de decisión
nombre		X	X	X	
nombre_pantalla		X	X	X	
localizacion_usuario	X	X	X	X	
url	X	X		X	
descripcion	X	X	X	X	X
cuenta_protegida				X	
cuenta_verificada	X			X	
numero_seguidores	X		X	X	X
numero_usuarios_seguidos			X	X	X
numero_listas		X		X	
numero_favoritos	X	X	X	X	X
numero_tweets_rts			X	X	X
fecha_creacion	X	X	X	X	X
url_banner_perfil	X	X		X	
url_imagen_perfil	X	X	X	X	X
perfil_por_defecto	X			X	
imagen_por_defecto				X	
Total de características	**10**	**10**	**10**	**17**	**7**

De esta forma, como se puede observar en la tabla anterior, la mayoría de las técnicas aplicadas dan una gran importancia especialmente a 6 variables en concreto: localizacion_usuario, descripcion, numero_seguidores, numero_favoritos, fecha_creacion y url_imagen_perfil. En contraposición, las variables menos seleccionadas por los métodos empleados resultan ser 5: cuenta_protegida, cuenta_verificada, numero_listas, perfil_por_defecto e imagen_por_defecto.

Así, tal y como se puede ver, **la técnica más restrictiva con un total de 7 características resulta ser "RFECV con árbol de decisión"**. De esta manera, **para poder trabajar con equidad, se procede a solicitar también al resto de técnicas las 7 variables más importantes, a fin de que cuenten todas con la misma cantidad de información**. Así pues, para reajustar la experimentación se volverán a ejecutar los fragmentos de código correspondientes a las mismas pero utilizando esta vez el valor 7 como número de características a seleccionar. De esta forma, se deberá de cambiar este valor numérico en las funciones "SelectKBest" y "RFE" utilizadas en el código. Asimismo, ya que el método "RFECV con regresión logística" no arrojaba ningún tipo de reducción de características, se descartará de la experimentación y se continuará únicamente con las otras 4 técnicas.

8.4. Reajuste de selección de características y nueva comparativa

En este apartado se procede a comparar las distintas selecciones de características que han realizado las técnicas aplicadas tras reajustar el valor de selección a 7 para tres de las mismas: "SelectKBest con el coeficiente de correlación ANOVA", "RFE con regresión logística" y "RFE con árbol de decisión". Además, para esta nueva comparativa se ha prescindido del método "RFECV con regresión logística", pues no reducía el número de atributos a seleccionar.

Tabla 8. Características seleccionadas para las distintas técnicas aplicadas tras el reajuste

Características	SelectKBest con el coeficiente de correlación ANOVA	RFE con regresión logística	RFE con árbol de decisión	RFECV con árbol de decisión
nombre				
nombre_pantalla				
localizacion_usuario	X	X		
url		X		
descripcion			X	X
cuenta_protegida				
cuenta_verificada	X			
numero_seguidores			X	X
numero_usuarios_seguidos			X	X
numero_listas		X		
numero_favoritos	X	X	X	X
numero_tweets_rts			X	X
fecha_creacion	X	X	X	X
url_banner_perfil	X	X		
url_imagen_perfil	X	X	X	X
perfil_por_defecto	X			
imagen_por_defecto				
Total de características	**7**	**7**	**7**	**7**

De esta forma, como se puede observar en la tabla anterior, la mayoría de las técnicas aplicadas dan una gran importancia especialmente a 3 variables en concreto: numero_favoritos, fecha_creacion y url_imagen_perfil. En contraposición, las variables menos seleccionadas por los métodos empleados resultan ser 5: url, cuenta_verificada, numero_listas y perfil_por_defecto. Además, existen varias características que no han sido seleccionadas por ninguna de las técnicas: nombre, nombre_pantalla, cuenta_protegida e imagen_por_defecto.

9. Clasificación binaria y comparativa

En este apartado se procede a explicar en detalle cómo se ha abordado el problema de clasificación, así como la implementación que se ha llevado a cabo para ello.

El problema principal por resolver es determinar si una cuenta de la red social Twitter, en base a sus características extraídas, resulta ser un bot (valor 0) o un humano (valor 1). De esta forma, puesto que la clasificación únicamente puede arrojar dos clases diferentes como resultado (0 o 1), se trata de un caso de **clasificación binaria.**

Para ello, se procede a desarrollar **modelos de perceptrones multicapa** o también conocidos como **MLP**, los cuales resultan ser redes neuronales totalmente conectadas. Estos modelos están compuestos de capas de nodos donde cada uno de ellos se encuentra conectado doblemente. De esta forma, cada nodo se encuentra conectado con las salidas procedentes de la capa predecesora y, a su vez, la salida de cada nodo se conecta con las entradas de los nodos de la siguiente capa [39]. Estos modelos serán aplicados sobre los conjuntos de datos resultantes del apartado anterior con sus respectivas reducciones (7 características), así como sobre el conjunto de datos original sin reducción (17 características).

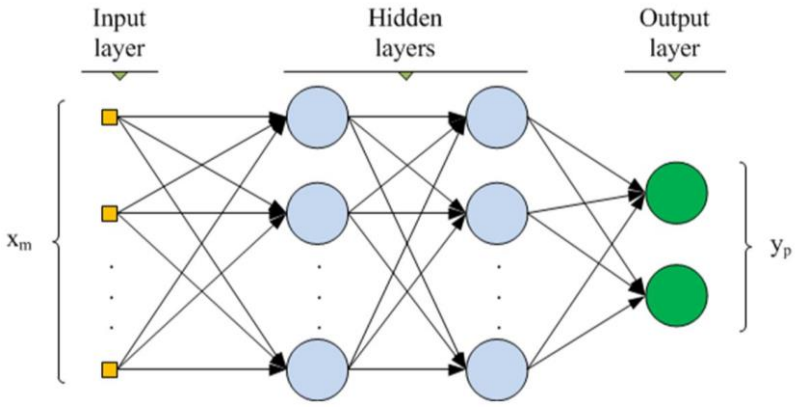


Ilustración 49. Estructura básica de un modelo de perceptrones multicapa o MLP (Fuente: https://www.intechopen.com/)

Así pues, para poder llevar a cabo el entrenamiento de los modelos y las predicciones de clasificación de los datos de usuarios, se utilizará la librería anteriormente mencionada **TensorFlow**.

9.1. Definición de los modelos

Para el presente trabajo se procede a diseñar modelos que presentarán una configuración inicial similar, pero algunos de ellos variarán ligeramente debido al uso de técnicas para mejorar el rendimiento que se explicarán más adelante.

Por consiguiente, la estructura básica presente en los modelos resulta ser la siguiente:

- **1 y 2 capas ocultas de nodos**, donde, en el caso de 1 capa, se hará uso de 100 nodos y, en el caso de 2 capas, se hará uso de 100 y 80.
- Las **capas ocultas** harán uso de las **funciones 'relu' (para la activación) y 'he_normal' (para la inicialización de pesos).**
- La **capa de salida** hará uso de la **función 'sigmoid' para la activación**, puesto que los modelos deben predecir la probabilidad de que se dé la clase 1, es decir, que se trate de un bot.

Así pues, en primer lugar, el código correspondiente a la definición del modelo con 1 capa oculta resultaría ser la siguiente:

```
#division de cada dataset en X e y
X = df.iloc[:,0:7] #df.iloc[:,0:17]
y = df.iloc[:,7].values #df.iloc[:,17].values

from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model

#separacion de los datos para entrenamiento y pruebas
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33)
print(X_train.shape, X_test.shape,y_train.shape, y_test.shape)

#cantidad de características de entrada
num_caracteristicas = X_train.shape[1]

#se define el modelo
model = Sequential()
#capa oculta 1
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))


model.add(Dense(1, activation='sigmoid'))

#grafico para ver la estructura de capas y conexiones del modelo definido
plot_model(model, 'grafico_modelo.png',show_shapes=True)
```

Código 16. Definición del modelo con 1 capa oculta

De esta forma, la estructura detallada de los modelos de 1 capa oculta presentaría la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

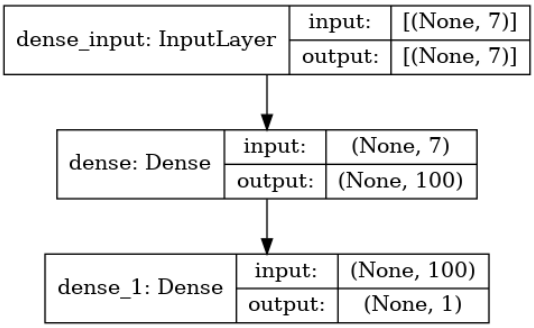


Ilustración 50. Gráfico de los modelos de 1 capa oculta con 7 nodos
(Fuente: propia)

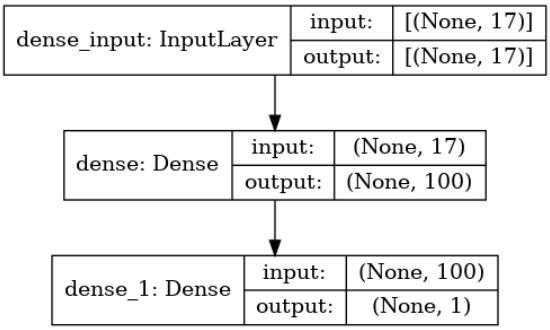


Ilustración 51. Gráfico de los modelos de 1 capa oculta con 17 nodos
(Fuente: propia)

En segundo lugar, el código correspondiente a la definición del modelo con 2 capas ocultas resultaría ser la siguiente:

```
#division de cada dataset en X e y
X = df.iloc[:,0:7] #df.iloc[:,0:17]
y = df.iloc[:,7].values #df.iloc[:,17].values

from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model

#separacion de los datos para entrenamiento y pruebas
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33)
print(X_train.shape, X_test.shape,y_train.shape, y_test.shape)

#cantidad de características de entrada
num_caracteristicas = X_train.shape[1]

#se define el modelo
model = Sequential()

#capa oculta 1
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))

#capa oculta 2
```

```
model.add(Dense(80, activation='relu',
kernel_initializer='he_normal'))

model.add(Dense(1, activation='sigmoid'))

#grafico para ver la estructura de capas y conexiones del modelo
definido
plot_model(model, 'grafico_modelo.png',show_shapes=True)
```

Código 17. Definición del modelo con 2 capas ocultas

De esta forma, la estructura detallada de los modelos de 2 capas ocultas presentaría la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

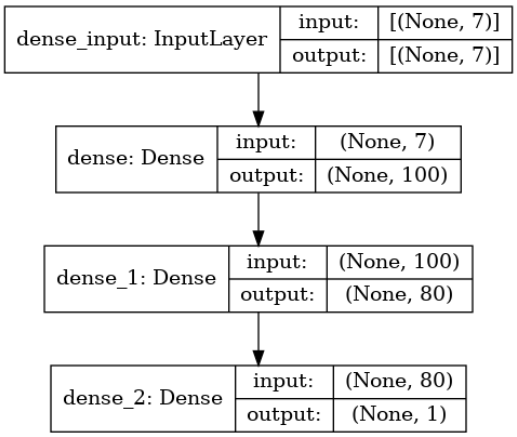


Ilustración 52. Gráfico de los modelos de 2 capas ocultas con 7 nodos (Fuente: propia)

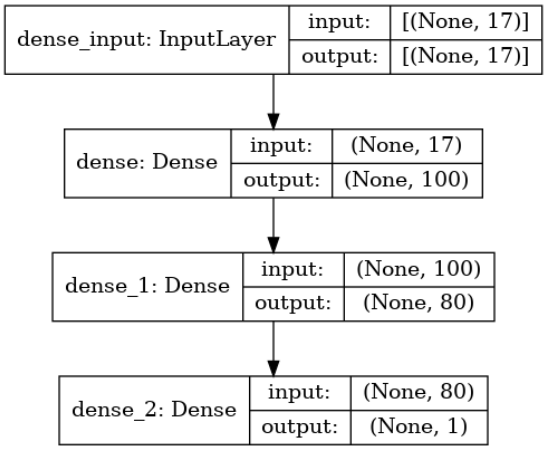


Ilustración 53. Gráfico de los modelos de 2 capas ocultas con 17 nodos (Fuente: propia)

9.2. Compilación y entrenamiento de los modelos

En este apartado se procede a explicar en detalle cómo se ha llevado a cabo la implementación correspondiente a la compilación y el entrenamiento de los distintos modelos.

En primer lugar, para la **compilación de los modelos** se ha escogido para su optimización el **algoritmo Adam ('adam') para el descenso de gradiente** y la función de pérdida logarítmica de **cruzamiento binario** ('binary_crossentropy') **para la pérdida**, especialmente utilizada para clasificaciones binarias.

Por consiguiente, la implementación correspondiente a la compilación de cada uno de los modelos resulta ser el siguiente:

```
model.compile(optimizer='adam',loss='binary_crossentropy',
metrics=['accuracy'])
```

Código 18. Compilación del modelo a entrenar

Seguidamente, se procede a explicar en detalle cómo se ha realizado la parte correspondiente a los entrenamientos de los distintos modelos. Para ello, se hace uso del método "fit" del modelo y se le indica por parámetro los valores de X e y de entrenamiento (X_train e y_train), así como el número de veces que se han de recorrer los datos de entrenamiento conocido como épocas ('epochs') y la división de validación ('validation_split') el cual permite indicar qué porcentaje de división correspondiente al conjunto de datos de entrenamiento se desea aplicar.

En este caso, se harán distintas combinaciones para cada modelo, probando con 100, 500 y 1000 épocas para el entrenamiento de cada modelo, con el fin de probar qué configuración de este parámetro logra arrojar mejores resultados de precisión y para el valor de la división de validación se usará el 20%. Asimismo, como el tiempo empleado en realizar cada entrenamiento es un factor también relevante para comparar rendimientos, se cronometrará la duración de cada uno de los entrenamientos.

Así pues, la implementación de la parte de entrenamiento resulta ser la siguiente:

```
from datetime import datetime

#hora de inicio del entrenamiento
inicio = datetime.now()

#entrenamiento del modelo – valores epochs 100/500/1000
modelo_entrenado = model.fit(X_train,y_train,epochs=NUMERO_EPOCHS,
validation_split=0.2)

#hora de finalización del entrenamiento y tiempo total transcurrido
fin = datetime.now()
tiempo_transcurrido = fin-inicio
```

```
minutos = divmod(tiempo_transcurrido.total_seconds(), 60)
print("Tiempo total empleado en el entrenamiento: ",minutos[0], 'minutos',
minutos[1], 'segundos')
```

Código 19. Implementación del entrenamiento del modelo y control del tiempo transcurrido

Por consiguiente, una vez se entrene el modelo, se procede a guardarlo en un fichero mediante el método "save" con el fin de poder hacer uso del mismo tantas veces se requiera sin necesidad de volver a ejecutar todo el proceso. Además, para consultar tanto la precisión como la pérdida del modelo se utilizará el método "evaluate" el cual recibirá los valores de X e y para pruebas (X_test e y_test).

```
#se guarda el modelo entrenado
model.save('nombre_del_modelo_entrenado.h5')
#evaluacion del modelo entrenado
perdida,precision = model.evaluate(X_test,y_test,verbose=0)
print('Prueba de precisión: %.3f' % precision)
```

Código 20. Guardado del modelo entrenado y evaluación de su precisión

9.3. Variaciones en los modelos: mejoras de rendimiento

En este apartado se procede a explicar en detalle algunas técnicas que se han utilizado para mejorar la actuación de los modelos definidos. Estas técnicas de mejora de rendimiento suelen, por norma general, centrarse en evitar el sobreentramiento y/o determinar el momento óptimo en el que debe detenerse el entrenamiento. Así pues, en este proyecto se utilizarán 3 técnicas de mejora de rendimiento por separado: abandono, normalización por lotes y parada anticipada.

9.3.1. Abandono

Esta técnica de mejora de rendimiento tiene como objetivo principal reducir el sobreajuste de los datos utilizados durante el entrenamiento. Para ello, a lo largo de todo el proceso de entrenamiento, la técnica ignora algunos resultados de cada capa de forma aleatoria, pudiendo reducir la cantidad de nodos que se perciben en cada capa, así como su conexión con la capa que le precede.

Así pues, para aplicar esta técnica, se debe añadir en la definición del modelo una capa "Dropout" antes de cada una de las capas sobre las cuales se desee reducir sus conexiones, indicándole por parámetro el porcentaje de entradas que se desea eliminar en cada actualización del modelo. En el caso de este proyecto se ha decidido añadir **una capa de abandono después de cada capa oculta** con un **porcentaje de eliminación del 20%.**

Así, por un lado, la definición de los modelos de 1 capa oculta aplicando esta técnica, presentan la siguiente implementación:

```
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Código 21. Definición del modelo con 1 capa oculta aplicando la técnica Dropout

De esta forma, la estructura detallada de los modelos de 1 capa oculta presentan la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

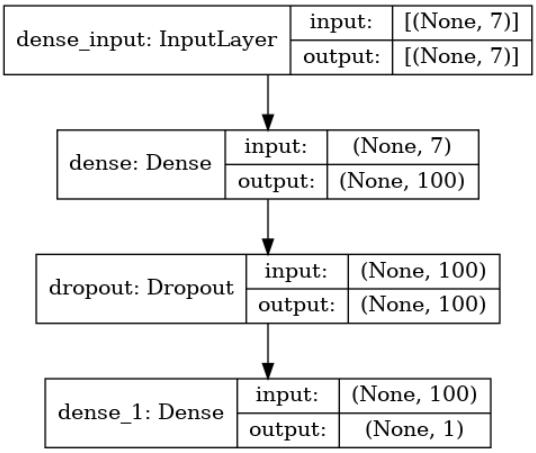


Ilustración 54. Gráfico de los modelos de 1 capa oculta con 7 nodos aplicando abandono (Fuente: propia)

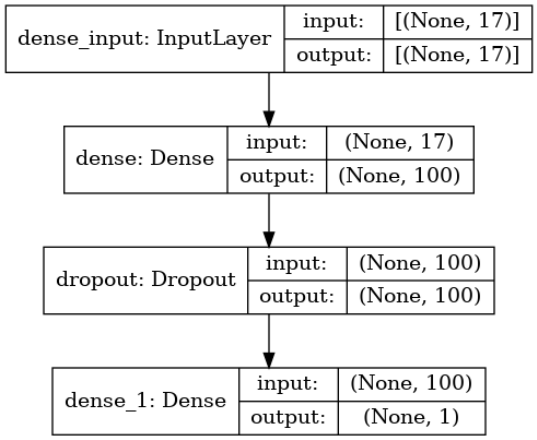


Ilustración 55. Gráfico de los modelos de 1 capa oculta con 17 nodos aplicando abandono (Fuente: propia)

Por otro lado, la definición de los modelos de 2 capas ocultas aplicando esta técnica, presentan la siguiente implementación:

```
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))
model.add(Dropout(0.2))
model.add(Dense(80, activation='relu', kernel_initializer='he_normal'))
```

```
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Código 22. Definición del modelo con 2 capas ocultas aplicando la técnica Dropout

De esta forma, la estructura detallada de los modelos de 2 capas ocultas presentan la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

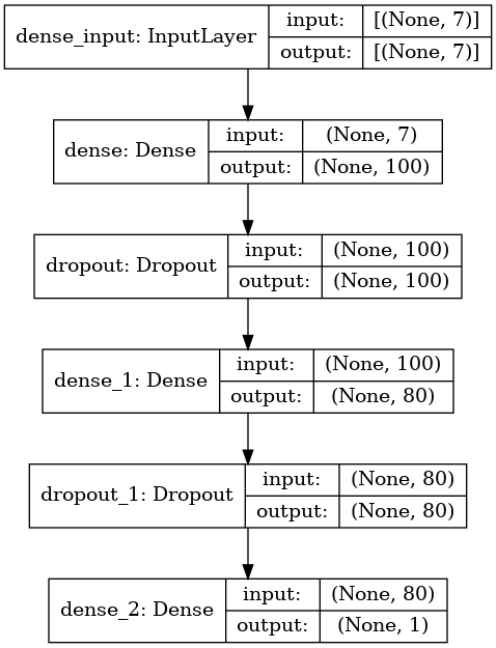


Ilustración 56. Gráfico de los modelos de 2 capas ocultas con 7 nodos aplicando abandono (Fuente: propia)

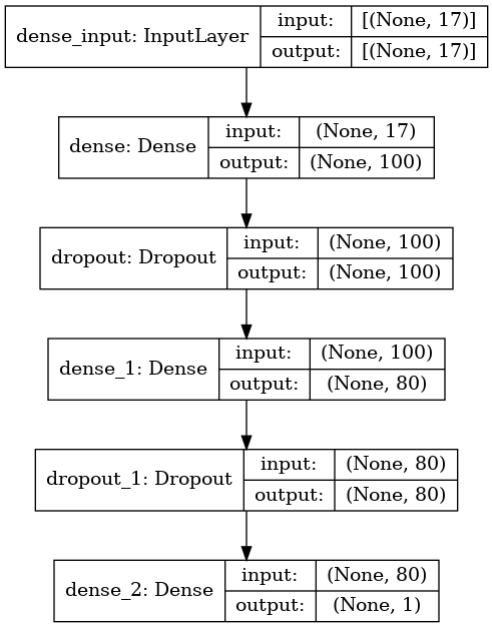


Ilustración 57. Gráfico de los modelos de 2 capas ocultas con 17 nodos aplicando abandono (Fuente: propia)

9.3.2. Normalización por lotes

Esta técnica de mejora de rendimiento tiene como objetivo principal acelerar el entrenamiento, estandarizando para ello las entradas de una capa para cada mini lote. Con ello, a lo largo de todo el proceso de entrenamiento, la técnica consigue minimizar de forma considerable la cantidad de épocas necesarias para el entrenamiento, así como estabilizar el aprendizaje.

Así pues, para aplicar esta técnica, se debe añadir en la definición del modelo una capa "BatchNormalization" antes de cada una de las capas sobre las cuales se desee estandarizar sus entradas. En el caso de este proyecto se ha decidido añadir **una capa de normalización después de cada capa oculta.**

Así, por un lado, la definición de los modelos de 1 capa oculta aplicando esta técnica, presentan la siguiente implementación:

```
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
```

Código 23. Definición del modelo con 1 capa oculta aplicando la técnica de Normalización por lotes

De esta forma, la estructura detallada de los modelos de 1 capa oculta presentan la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

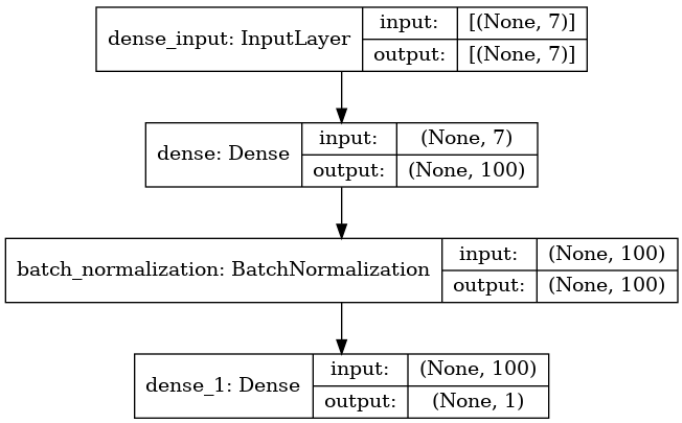


Ilustración 58. Gráfico de los modelos de 1 capa oculta con 7 nodos aplicando normalización por lotes (Fuente propia)

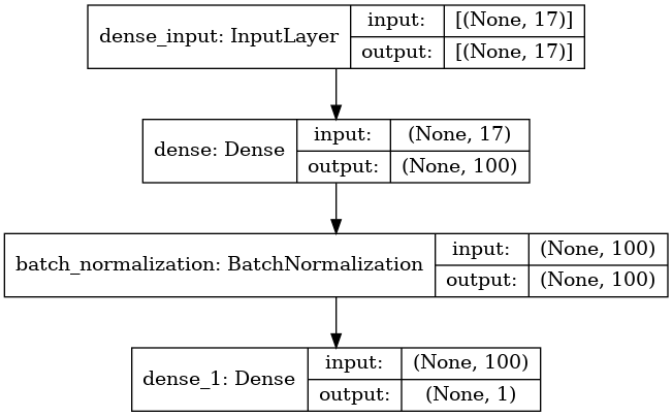


Ilustración 59. Gráfico de los modelos de 1 capa oculta con 17 nodos aplicando normalización por lotes (Fuente propia)

Por otro lado, la definición de los modelos de 2 capas ocultas aplicando esta técnica, presentan la siguiente implementación:

```
model.add(Dense(100, activation='relu',
kernel_initializer='he_normal',input_shape=(num_caracteristicas,)))
model.add(BatchNormalization())
model.add(Dense(80, activation='relu', kernel_initializer='he_normal'))
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
```

Código 24. Definición del modelo con 2 capas ocultas aplicando la técnica Normalización por lotes

De esta forma, la estructura detallada de los modelos de 2 capas ocultas presentan la siguiente estructura visual para 7 y 17 características de entrada respectivamente:

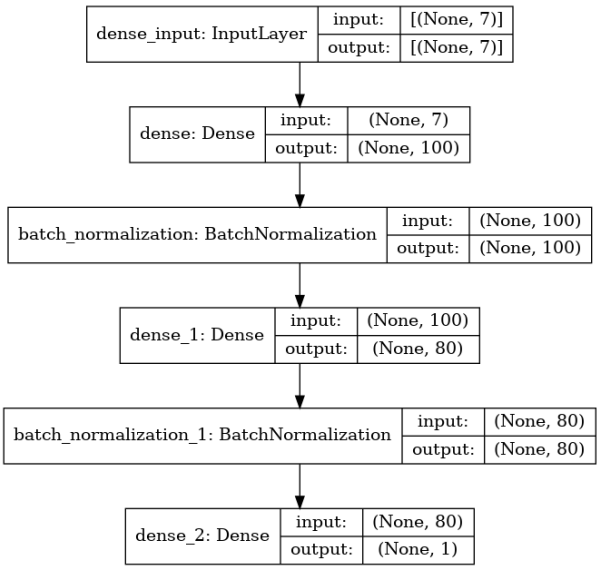


Ilustración 60. Gráfico de los modelos de 2 capas ocultas con 7 nodos aplicando normalización por lotes (Fuente propia)

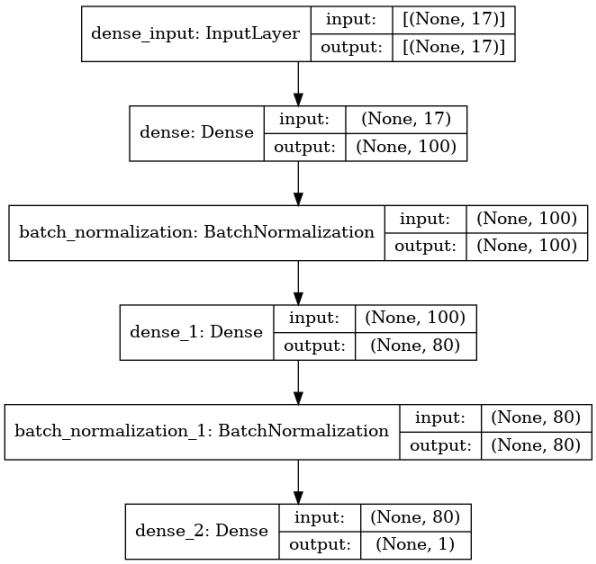


Ilustración 61. Gráfico de los modelos de 2 capas ocultas con 17 nodos aplicando normalización por lotes (Fuente propia)

9.3.3. Parada anticipada

Esta técnica de mejora de rendimiento tiene como objetivo principal detener el entrenamiento en el instante preciso con el fin de evitar el entrenamiento en exceso o la falta de este. Para ello, a lo largo de todo el proceso de entrenamiento, la técnica monitoriza la pérdida de los datos de entrenamiento y los datos de validación, de tal forma que en cuanto se observan signos de sobreajuste el proceso se detiene.

Así pues, para aplicar esta técnica, no será necesario modificar la definición de los modelos, pues únicamente será necesario definir un EarlyStopping sobre el que configurar qué medida se va a monitorizar (en este caso el vapor de pérdida 'val_loss') junto con la cantidad de épocas de sobreentrenamiento ('patience') que se deben de esperar para detener el proceso de entrenamiento. En este caso como valor de espera se utilizará 5, presentando así la siguiente implementación:

```
from datetime import datetime

#hora de inicio del entrenamiento
inicio = datetime.now()

#valor de espera para la parada anticipada
es = EarlyStopping(monitor='val_loss', patience=5)

#entrenamiento del modelo - valores epochs 100/500/1000
modelo_entrenado = model.fit(X_train,y_train,epochs=NUMERO_EPOCHS,
validation_split=0.2,callbacks=[es])
```

```
#hora de finalización del entrenamiento y tiempo total transcurrido
fin = datetime.now()
tiempo_transcurrido = fin-inicio
minutos = divmod(tiempo_transcurrido.total_seconds(), 60)
print("Tiempo total empleado en el entrenamiento: ",minutos[0], 'minutos',
minutos[1], 'segundos')
```

Código 25. Aplicación de la técnica de parada anticipada sobre el entrenamiento del modelo

9.4. Modelos entrenados y sus resultados

En este apartado se procede a explicar en detalle las distintas configuraciones de modelos que se han entrenado, así como los tiempos empleados para el entrenamiento y la precisión final obtenida en cada caso. Así, para las distintas configuraciones de modelos entrenados se han contemplado los siguientes aspectos:

- Se han llevado a cabo entrenamientos **tanto con técnicas de mejora de rendimiento (abandono, normalización por lotes y parada anticipada) como sin mejoras.**
- Se han aplicado distintos valores de épocas: **100, 500 y 1000.**
- Se han entrenado modelos tanto de **1 como de 2 capas.**

9.4.1. Modelos entrenados con las características seleccionadas con ANOVA

En esta sección se muestran los resultados obtenidos para cada una de las configuraciones de modelos entrenadas utilizando el conjunto de datos resultante tras aplicar la reducción de características de ANOVA.

1 CAPA

Por un lado, los resultados de ejecución para los **modelos entrenados con 1** capa utilizando este dataset reducido resultan ser los siguientes:

Tabla 9. Comparativa de los entrenamientos de ANOVA con 1 capa

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	77,7%	4’ 38’’
500	500	-	76,9%	27’ 16’’
1000	1000	-	77,8%	51’ 34’’
100	100	Abandono	60,9%	5’ 3’’
500	500	Abandono	61,2%	26’ 43’’
1000	1000	Abandono	61,8%	53’ 20’’
100	**100**	**Normalización por lotes**	**78,9%**	**5’ 32’’**
500	500	Normalización por lotes	78,9%	30’ 30’’
1000	**1000**	**Normalización por lotes**	**80,3%**	**1h 6’ 45’’**
100	37	Parada anticipada	75,3%	1’ 38’’
500	24	Parada anticipada	74,7%	1’ 4’’
1000	21	Parada anticipada	71%	51’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 1000 épocas**, alcanzando una precisión del 80,3% tras un entrenamiento de más de una hora.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 78,9% en poco más de 5 minutos.

2 CAPAS

Por otro lado, los resultados de ejecución para los **modelos entrenados con 2 capas** utilizando este dataset reducido resultan ser los siguientes:

Tabla 10. Comparativa de los entrenamientos de ANOVA con 2 capas

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	80,1%	5’ 28’’
500	500	-	81%	31’ 42’’
1000	1000	-	61,8%	1h 58’’
100	100	Abandono	60,3%	5’ 54’’
500	500	Abandono	61%	32’40’’
1000	1000	Abandono	61%	1h 9’ 20’’
100	**100**	**Normalización por lotes**	**82,1%**	**8’ 19’’**
500	500	Normalización por lotes	82,3%	45’ 14’’
1000	**1000**	**Normalización por lotes**	**82,8%**	**1h 25’ 13’’**
100	34	Parada anticipada	77%	1’ 40’’
500	51	Parada anticipada	79,2%	2’ 43’’
1000	61	Parada anticipada	80,2%	3’ 23’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 1000 épocas**, alcanzando una precisión del 82,8% tras un entrenamiento de casi una hora y media.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 82,1% en poco más de 8 minutos.

9.4.2. Modelos entrenados con las características seleccionadas con RFE Regresión Logística

En esta sección se muestran los resultados obtenidos para cada una de las configuraciones de modelos entrenadas utilizando el conjunto de datos resultante tras aplicar la reducción de características de RFE con Regresión Logística.

1 CAPA

Por un lado, los resultados de ejecución para los **modelos entrenados con 1** capa utilizando este dataset reducido resultan ser los siguientes:

Tabla 11. Comparativa de los entrenamientos de RFE Regresión Logística con 1 capa

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	76,5%	4’ 58’’
500	500	-	79,7%	26’ 14’’
1000	1000	-	75,9%	53’ 29’’
100	100	Abandono	73%	5’ 13’’
500	500	Abandono	74,1%	25’ 15’’
1000	1000	Abandono	60,6%	53’ 24’’
100	**100**	**Normalización por lotes**	**80,9%**	**6’ 29’’**
500	500	Normalización por lotes	81,5%	32’ 4’’
1000	**1000**	**Normalización por lotes**	**82,1%**	**1h 3’ 12’’**
100	32	Parada anticipada	75,3%	1’ 23’’
500	16	Parada anticipada	70,1%	1’ 41’’
1000	29	Parada anticipada	76,4%	1’ 7’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 1000 épocas**, alcanzando una precisión del 82,1% tras un entrenamiento de poco más de una hora.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 80,9% en poco más de 6 minutos.

2 CAPAS

Por otro lado, los resultados de ejecución para los **modelos entrenados con 2 capas** utilizando este dataset reducido resultan ser los siguientes:

Tabla 12. Comparativa de los entrenamientos de RFE Regresión Logística con 2 capas

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	82,1%	6’ 4’’
500	500	-	81,8%	33’ 28’’
1000	1000	-	82,5%	1h 3’ 39’’
100	100	Abandono	61,1%	6’ 48’
500	500	Abandono	60,8%	34’ 16’’
1000	1000	Abandono	60,5%	1h 9’ 19’’
100	**100**	**Normalización por lotes**	**84%**	**7’ 45’’**
500	**500**	**Normalización por lotes**	**84,4%**	**42’ 56’’**
1000	1000	Normalización por lotes	84,1%	1h 31’ 47’’
100	35	Parada anticipada	77,8%	1’ 49’’
500	43	Parada anticipada	77,6%	2’ 23’’
1000	56	Parada anticipada	79,1%	3’ 9’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 500 épocas**, alcanzando una precisión del 84,4% tras un entrenamiento de casi 43 minutos.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 84% en poco más de 7 minutos.

9.4.3. Modelos entrenados con las características seleccionadas con RFE Árbol de decisión / RFECV Árbol de decisión

En esta sección se muestran los resultados obtenidos para cada una de las configuraciones de modelos entrenadas utilizando el conjunto de datos resultante tras aplicar la reducción de características de RFE o RFECV con Árbol de decisión.

1 CAPA

Por un lado, los resultados de ejecución para los **modelos entrenados con 1** capa utilizando este dataset reducido resultan ser los siguientes:

Tabla 13. Comparativa de los entrenamientos de RFE/RFECV Árbol de decisión con 1 capa

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	79,4%	5’ 6’’
500	500	-	72,1%	24’ 42’’
1000	1000	-	61,1%	48’ 52’’
100	100	Abandono	61,1%	5’ 15’’
500	500	Abandono	60,9%	27’
1000	1000	Abandono	61,5%	51’ 14’’
100	**100**	**Normalización por lotes**	**83,8%**	**6’ 42’’**
500	500	Normalización por lotes	82,3%	32’
1000	**1000**	**Normalización por lotes**	**84,6%**	**1h 8’ 24’’**
100	23	Parada anticipada	77%	58’’
500	13	Parada anticipada	76,1%	31’’
1000	17	Parada anticipada	75,9%	39’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 1000 épocas**, alcanzando una precisión del 84,6% tras un entrenamiento de más de una hora.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 83,8% en poco más de 6 minutos.

2 CAPAS

Por otro lado, los resultados de ejecución para los **modelos entrenados con 2 capas** utilizando este dataset reducido resultan ser los siguientes:

Tabla 14. Comparativa de los entrenamientos de RFE/RFECV Árbol de decisión con 2 capas

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	84%	5’ 20’’
500	500	-	86,5%	30’ 19’’
1000	1000	-	60,8%	1h 5’ 13’’
100	100	Abandono	61,1%	6’ 16’’
500	500	Abandono	60,7%	34’ 53’’
1000	1000	Abandono	60,8%	1h 6’ 37’’
100	**100**	**Normalización por lotes**	**86,5%**	**7’ 33’’**
500	500	Normalización por lotes	85,9%	45’ 2’’
1000	**1000**	**Normalización por lotes**	**87,8%**	**1h 29’ 45’’**
100	36	Parada anticipada	78,3%	1’ 57’’
500	28	Parada anticipada	75,8%	1’ 28’’
1000	43	Parada anticipada	80,2%	2’ 23’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 1000 épocas**, alcanzando una precisión del 87,8% tras un entrenamiento de casi una hora y media.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 86,5% en poco más de 7 minutos.

9.4.4. Modelos entrenados con las características sin reducir

En esta sección se muestran los resultados obtenidos para cada una de las configuraciones de modelos entrenadas utilizando el conjunto de datos sin aplicar ningún tipo de reducción de características.

1 CAPA

Por un lado, los resultados de ejecución para los **modelos entrenados con 1** capa utilizando el dataset sin reducciones resultan ser los siguientes:

Tabla 15. Comparativa de los entrenamientos de las características originales con 1 capa

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	70%	4’ 48’’
500	500	-	60,8%	25’ 58’’
1000	1000	-	79,3%	54’ 59’’
100	100	Abandono	61,2%	5’ 28’’
500	500	Abandono	60,8%	20’ 20’’

1000	1000	Abandono	60,5%	59’ 16’’
100	**100**	**Normalización por lotes**	**84,7%**	**6’ 4’’**
500	**500**	**Normalización por lotes**	**85,8%**	**32’ 56’’**
1000	1000	Normalización por lotes	82%	1h 8’ 25’’
100	17	Parada anticipada	75,9%	40’’
500	18	Parada anticipada	69,3%	43’’
1000	16	Parada anticipada	75,5%	39’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 500 épocas**, alcanzando una precisión del 85,9% tras un entrenamiento de más de poco más de media hora.

Sin embargo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser la combinación de **normalización por lotes y 100 épocas**, pues ha alcanzado un 84,7% en poco más de 6 minutos.

2 CAPAS

Por otro lado, los resultados de ejecución para los **modelos entrenados con 2 capas** utilizando el dataset sin reducciones resultan ser los siguientes:

Tabla 16. Comparativa de los entrenamientos de las características originales con 2 capas

Epochs configurados	Epochs transcurridos	Mejora de rendimiento aplicada	Precisión obtenida %	Duración del entrenamiento
100	100	-	60,9%	5’ 46’’
500	500	-	60,8%	31’ 18’’
1000	1000	-	61,1%	1h 10’ 7’’
100	100	Abandono	60,8%	6’ 10’’
500	500	Abandono	60,9%	33’ 8’’
1000	1000	Abandono	60,7%	1h 10’ 35’’
100	**100**	**Normalización por lotes**	**87,2%**	**8’ 19’’**
500	500	Normalización por lotes	86,5%	45’ 9’’
1000	1000	Normalización por lotes	84,9%	1h 24’ 44’’
100	48	Parada anticipada	76,6%	2’ 42’’
500	47	Parada anticipada	76,8%	2’ 31’’
1000	65	Parada anticipada	82,6%	3’ 31’’

En este caso la configuración que **mayor precisión ha obtenido** resulta ser la combinación de **normalización por lotes y 100 épocas**, alcanzando una precisión del 87,2% tras un entrenamiento de más de poco más de 8 minutos.

Asimismo, la configuración que ha logrado el **mejor equilibrio** respecto a la precisión obtenida y la duración del entrenamiento resulta ser **la misma combinación que ha registrado la mayor precisión.**

9.4.5. Análisis de los resultados obtenidos y reajustes

En este apartado se procede a realizar un análisis comparativo entre las mejores combinaciones de entrenamiento comentadas con anterioridad para cada uno de los distintos datasets.

Tabla 17. Comparativa de los entrenamientos con mejor precisión y equilibrio

Técnica de selección de entradas	Mejora rendimiento	Capas	Epochs	Precisión obtenida en test	Tiempo del entrenamiento
ANOVA	Normalización Lotes	1	100	78,9%	5’ 32’’
ANOVA	Normalización Lotes	1	1000	80,3%	1h 6’ 45’’
ANOVA	Normalización Lotes	2	100	82,1%	8’ 19’’
ANOVA	Normalización Lotes	2	1000	82,8%	1h 25’ 13’’
RFE RL	Normalización Lotes	1	100	80,9%	6’ 29’’
RFE RL	Normalización Lotes	1	1000	82,1%	1h 3’ 12’’
RFE RL	Normalización Lotes	2	100	84%	7’ 45’’
RFE RL	Normalización Lotes	2	500	84,4%	42’ 56’’
RFE/RFECVAD	Normalización Lotes	1	100	83,8%	6’ 42’’
RFE/RFECVAD	Normalización Lotes	1	1000	84,6%	1h 8’ 24’’
RFE/RFECVAD	**Normalización Lotes**	**2**	**100**	**86,5%**	**7’ 33’’**
RFE/RFECVAD	**Normalización Lotes**	**2**	**1000**	**87,8%**	**1h 29’ 45’’**
-	Normalización Lotes	1	100	84,7%	7’ 4’’
-	**Normalización Lotes**	**1**	**500**	**85,8%**	**32’ 56’’**
-	**Normalización Lotes**	**2**	**100**	**87,2%**	**8’ 19’’**

➔ ***Tiempos de entrenamiento y precisión***

Tal y como se puede apreciar en la tabla anterior, generalmente aquellos **modelos que utilizan 17 características de entrada presentan entrenamientos con tiempos ligeramente mayores** que en el caso de los modelos que utilizan 7 entradas. Además, tanto los **modelos con 17 características** de entrada **como los de 7 obtienen precisiones muy similares**, hecho que evidencia el **correcto funcionamiento de la selección de características** ya que los **7 atributos de entrada resultan ser suficientemente decisivos** para lograr el correcto entrenamiento de los modelos.

➔ ***Modelos seleccionados***

De esta forma, tal y como se puede apreciar en la tabla comparativa, los 4 entrenamientos que mayor precisión han conseguido emplearon el dataset con **reducción de características RFE/RFECV Árbol de Decisión y el dataset completo** sin reducción.

Así, en primer lugar, los mejores entrenamientos utilizando el **dataset con reducción de carácterísticas RFE/RFECV Árbol de Decisión**, fueron aquellos **con 2 capas junto con 100 y 1000 épocas**. En segundo lugar, los mejores entrenamientos utilizando el **dataset completo sin reducir**, han resultado ser de **1 capa junto con 500 épocas y de 2 capas junto con 100 épocas.** Sin embargo, para evaluar la calidad de estos 4 entrenamientos es necesario analizar la evolución de la precisión y la pérdida durante sus ejecuciones con el fin de valorar si necesitan algún tipo de reajuste o son correctos.

Para ello, será necesario obtener las gráficas que muestren los historiales de precisión y pérdida del modelo a lo largo de su entrenamiento, utilizando para ello la siguiente implementación:

```
from matplotlib import pyplot as plt
#evolución de la precisión
plt.plot(modelo_entrenado.history['accuracy'])
plt.plot(modelo_entrenado.history['val_accuracy'])
plt.title('Precisión del modelo')
plt.ylabel('precisión')
plt.xlabel('epoch')
plt.legend(['train','test'])
plt.show()
#evolución de la pérdida
plt.plot(modelo_entrenado.history['loss'])
plt.plot(modelo_entrenado.history['val_loss'])
plt.title('Pérdida del modelo')
plt.ylabel('pérdida')
plt.xlabel('epoch')
plt.legend(['train','test'])
plt.show()
```

Código 26. Obtención de las gráficas con la historia de precisión y pérdida a lo largo del entrenamiento

Por consiguiente, una vez obtenidas estas gráficas, será necesario buscar signos en las mismas que puedan indicar si existen problemas de overfitting o underfitting en el entrenamiento del modelo e intentar minimizar estos aspectos cuanto sea posible reajustando las configuraciones de los modelos. Así, para entender mejor en qué consisten ambos conceptos, se procede a explicarlos brevemente [40]:

- **Overfitting** (sobreajuste): tiene lugar cuando el modelo deja de aprender y memoriza los datos de aprendizaje. Así, este hecho impacta gravemente en el rendimiento del modelo, ya que en cuando recibe datos nuevos y desconocidos no

tiene capacidad para generalizar. Este aspecto se suele apreciar cuando los valores de pérdida de validación son altos y los de entrenamiento bajos.

- **Underfitting** (infrajuste): tiene lugar cuando el modelo es incapaz de aprender con los datos de entrenamiento aportados ya que resultan ser insuficientes y, por tanto, no es capaz de determinar ningún patrón. Este aspecto se suele apreciar cuando los valores de pérdida de entrenamiento y validación son altos.

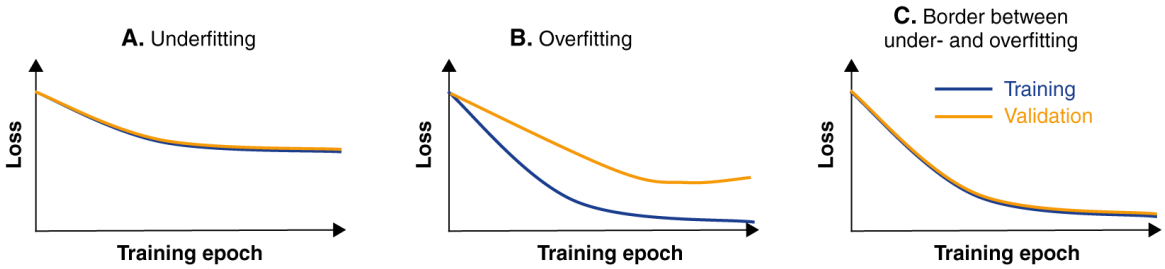


Ilustración 62. Tipos de entrenamientos según su pérdida
(Fuente: https://livebook.manning.com/)

De esta forma, se procede a obtener ambas gráficas para los 4 modelos de entrenamiento que presentaron mayores porcentajes de precisión con el fin de reajustar los modelos en los casos que fuera necesario.

RFE con árbol de decisión, normalización por lotes, 2 Capas y 100 epoch

Para la **primera ejecución** del entrenamiento de este modelo, utilizando **normalización por lotes, 2 capas y 100 epoch, se obtiene una precisión del 86,5%.** Seguidamente, se procede a visualizar sus gráficas correspondientes a la evolución de su precisión y su pérdida a lo largo del entrenamiento:

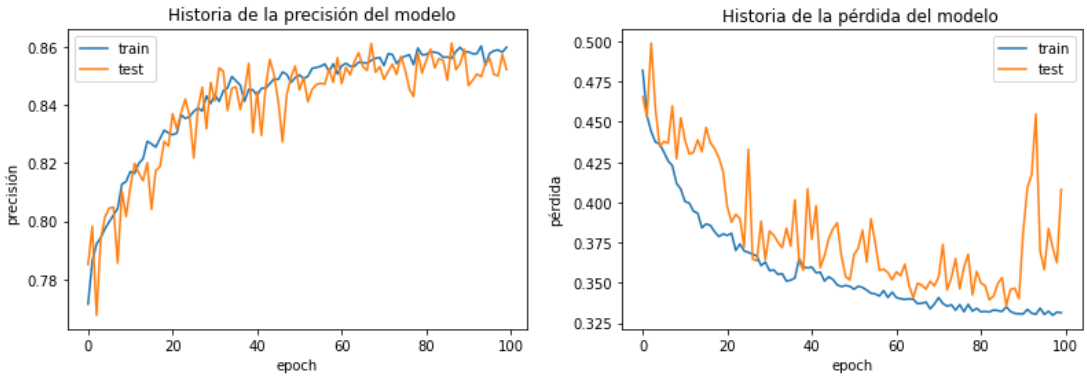


Ilustración 63. Gráficas de precisión y pérdida del modelo RFE 2C 100 – Ejecución 1
(Fuente propia)

Así, por un lado, se observa que **la evolución de la precisión del modelo es correcta,** ya que crece a medida que van transcurriendo las épocas configuradas para la ejecución. Por otro lado,

en la gráfica correspondiente a la evolución de la pérdida del modelo, se observa que **a partir de los 90 epoch empieza a apreciarse overfitting**, pues **los valores de pérdida de validación o test son altos y los de entrenamiento bajos.**

Por consiguiente, **para intentar minimizar este hecho**, se procede a ejecutar nuevamente el entrenamiento pero **esta vez bajando a 80 epoch** con el fin de intentar corregir el overfitting.

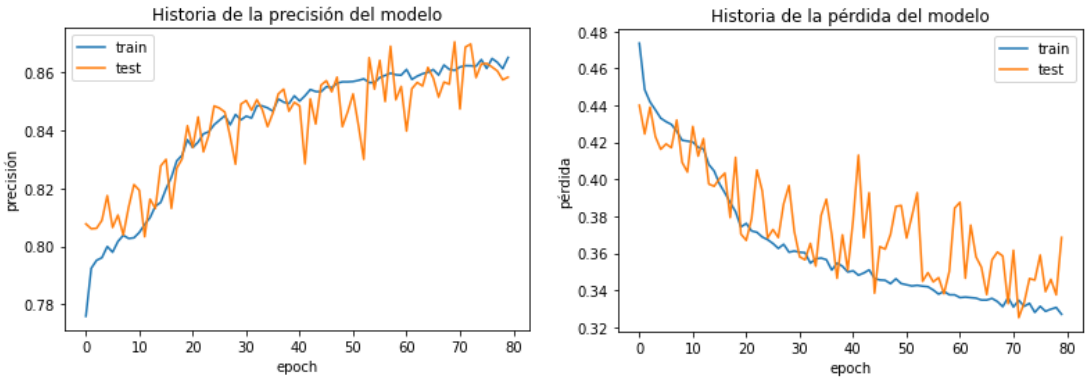


Ilustración 64. Gráficas de precisión y pérdida del modelo RFE 2C 100 – Ejecución 2 (Fuente propia)

De esta forma, tras este primer reajuste, la **precisión resultante disminuye a 86,1% con respecto a la inicial** y se observa nuevamente que la tendencia de la misma es correcta a lo largo del entrenamiento. Sin embargo, **los valores de pérdida de validación o test empiezan a subir en la parte final del entrenamiento.** Así pues, para intentar reducir este **comienzo de overfitting**, se procede a aumentar el conjunto de datos de aprendizaje pasando del 66,66% al 80%, con el fin de proporcionar al modelo más información para que mejore su entrenamiento.

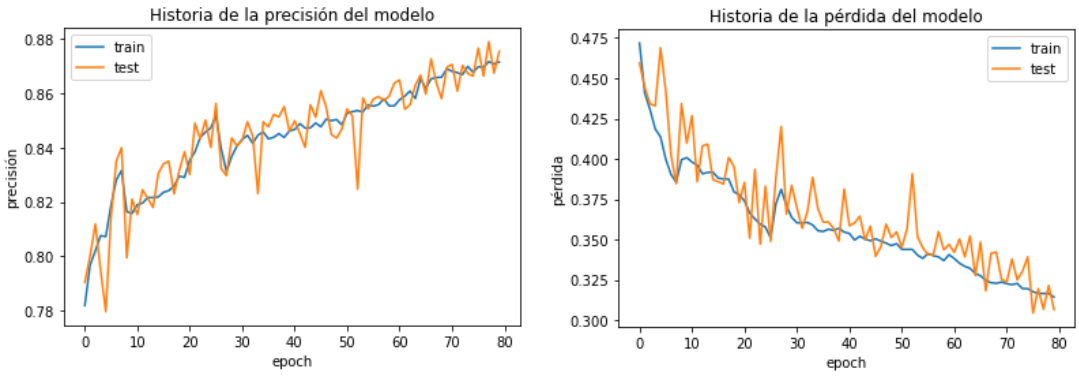


Ilustración 65. Gráficas de precisión y pérdida del modelo RFE 2C 100 – Ejecución 3 (Fuente propia)

Por consiguiente, tras el segundo reajuste, la **precisión resultante mejora a 86,7% con respecto a las anteriores** y se observa nuevamente que la tendencia de la misma se mantiene correcta. Asimismo, se observa que **los valores de pérdida de validación o test no presentan ninguna subida importante y van muy parejos con los valores de entrenamiento**, quedando por tanto el problema del **overfitting controlado** para este modelo.

RFE con árbol de decisión, normalización por lotes, 2 Capas y 1000 epoch

Para la **primera ejecución** del entrenamiento de este modelo, utilizando **normalización por lotes, 2 capas y 1000 epoch, se obtiene una precisión del 87,8%.** Seguidamente, se procede a visualizar sus gráficas correspondientes a la evolución de su precisión y su pérdida a lo largo del entrenamiento:

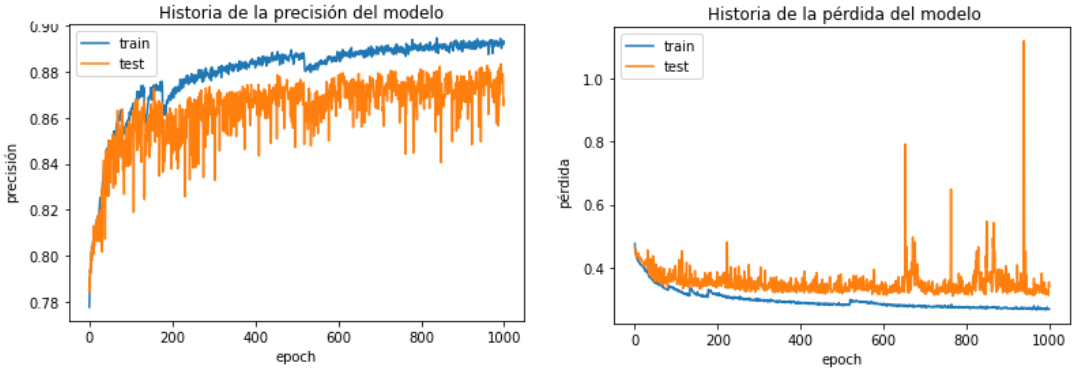


Ilustración 66. Gráficas de precisión y pérdida del modelo RFE 2C 1000 – Ejecución 1 (Fuente propia)

Así, por un lado, se observa que **la evolución de la precisión tiende a crecer a lo largo del entrenamiento, pero existe bastante distancia entre los valores de entrenamiento y validación, especialmente a partir de los 200 epoch**. Por otro lado, en la gráfica correspondiente a la evolución de la pérdida del modelo, se observa que aproximadamente **a partir de los 200 epoch empieza a apreciarse overfitting**, pues **los valores de pérdida de validación o test comienzan a crecer y a tomar distancia con respecto a los valores de entrenamiento que se mantienen bajos.** Por consiguiente, **para intentar minimizar este hecho**, se procede a ejecutar nuevamente el entrenamiento, pero **esta vez bajando a 200 epoch** con el fin de intentar corregir el overfitting.

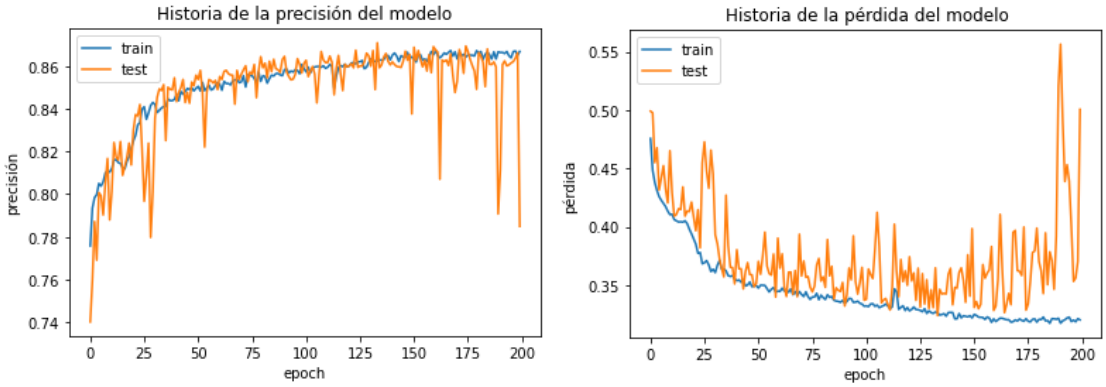


Ilustración 67. Gráficas de precisión y pérdida del modelo RFE 2C 1000 – Ejecución 2 (Fuente propia)

De esta forma, tras este primer reajuste, la **precisión resultante disminuye a 78,7% con respecto a la inicial,** sin embargo, la precisión para validación o test ahora sí presenta valores muy similares a los de entrenamiento y no existe tanta distancia entre ambos como pasaba en la ejecución anterior. Sin embargo, hacia los últimos epoch del entrenamiento, la precisión experimenta algunos valores atípicos bastante bajos. Asimismo, **los valores de pérdida de validación o test presentan ahora valores más cercanos a los de entrenamiento** aunque **suben ligeramente en la parte final** del entrenamiento, especialmente **a partir de los 180 epoch.** Así pues, para intentar reducir este **overfitting**, se procede a ejecutar nuevamente el entrenamiento pero **esta vez bajando a 180 epoch** con el fin de minimizar este hecho.

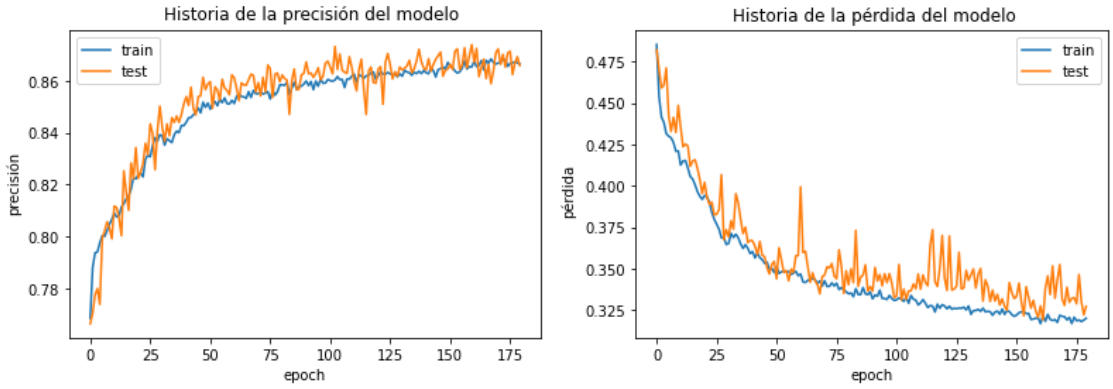


Ilustración 68. Gráficas de precisión y pérdida del modelo RFE 2C 1000 – Ejecución 3 (Fuente propia)

Por consiguiente, tras el tercer reajuste, la **precisión resultante mejora a 86,6% con respecto a la ejecución anterior** y se aprecia que la tendencia de la misma ahora es correcta. Asimismo, se observa que **los valores de pérdida de validación o test presentan algunas subidas puntuales especialmente en la segunda mitad del entrenamiento**. Así pues, para intentar reducir este ligero **comienzo de overfitting**, se procede a aumentar el conjunto de datos de aprendizaje pasando del 66,66% al 80%, con el fin de proporcionar al modelo más información para que mejore su entrenamiento.

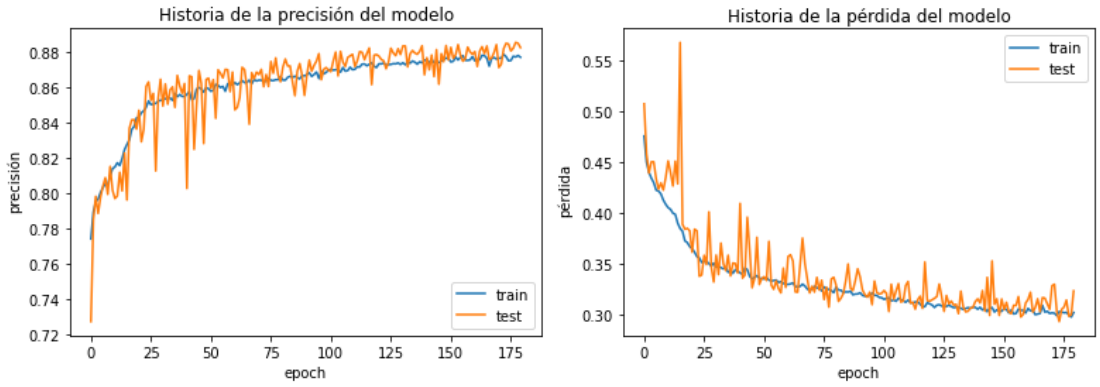


Ilustración 69. Gráficas de precisión y pérdida del modelo RFE 2C 1000 – Ejecución 4 (Fuente propia)

Finalmente, tras el cuarto reajuste, la **precisión resultante mejora a 87,8% con respecto a la ejecución anterior** y se observa que la tendencia de la misma se mantiene correcta. Tal y como se puede apreciar si se compara con la precisión inicial de 87,8%, ambas coinciden. Asimismo, se percibe que **los valores de pérdida de validación o test ya no presentan ninguna subida importante en la segunda mitad del entrenamiento.** De esta forma, **sus valores van muy parejos con los de entrenamiento**, **exceptuando algún valor atípico puntual** que tiene lugar durante los primeros epoch que al ser un caso aislado no cobra mayor trascendencia. Por tanto, se da por controlado el problema del inicio de **overfitting** para este modelo.

Todas las características, normalización por lotes, 1 capa y 500 epoch

Para la **primera ejecución** del entrenamiento de este modelo, utilizando **normalización por lotes, 1 capa y 500 epoch, se obtiene una precisión del 85,8%.** Seguidamente, se procede a visualizar sus gráficas correspondientes a la evolución de su precisión y su pérdida a lo largo del entrenamiento:

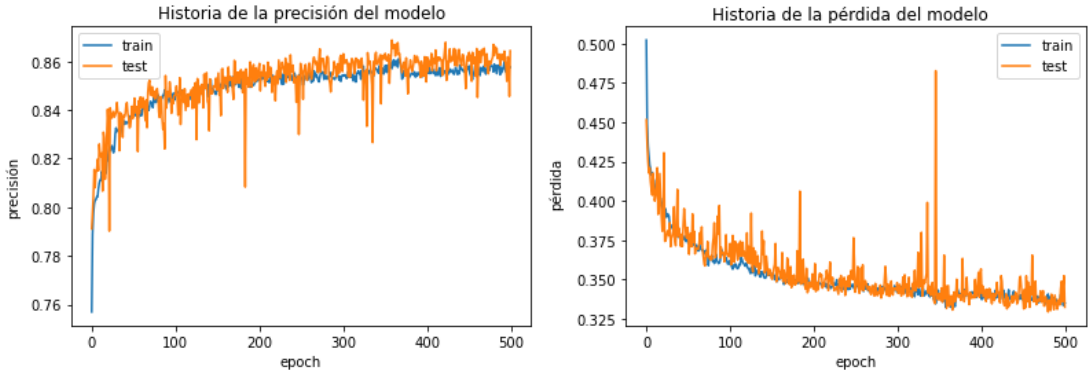


Ilustración 70. Gráficas de precisión y pérdida del modelo Todas las características 1C 500 – Ejecución 1 (Fuente propia)

Así, por un lado, se observa que **la tendencia de la precisión del modelo es correcta,** ya que crece a medida que van transcurriendo las épocas configuradas para la ejecución**, a excepción** de **algunos valores atípicos** que tienen lugar **entre los 150 y 400 epoch**. Por otro lado, en la gráfica correspondiente a la evolución de la pérdida del modelo, **no se observan signos ni de overfitting ni de underfitting en su tendencia**, pues los valores de pérdida tanto de entrenamiento como de validación o test resultan muy parejos entre sí, **exceptuando** también en este caso **algunos valores atípicos** que tienen lugar **entre los 150 y 400 epoch**.

Estos **valores puntuales tan bajos de precisión y tan altos en la pérdida que generan mucho ruido en las gráficas mostradas**, se entiende que son esperables debido a la **heterogeneidad de los datos que recibe el modelo como entrada**, pues **habrá registros** de usuarios **que el modelo entiende de forma clara y otros que le resulten más raros** o complejos de catalogar.

Todas las características, normalización por lotes, 2 capas y 100 epoch

Para la **primera ejecución** del entrenamiento de este modelo, utilizando **normalización por lotes, 2 capas y 100 epoch, se obtiene una precisión del 87,2%.** Seguidamente, se procede a visualizar sus gráficas correspondientes a la evolución de su precisión y su pérdida a lo largo del entrenamiento:

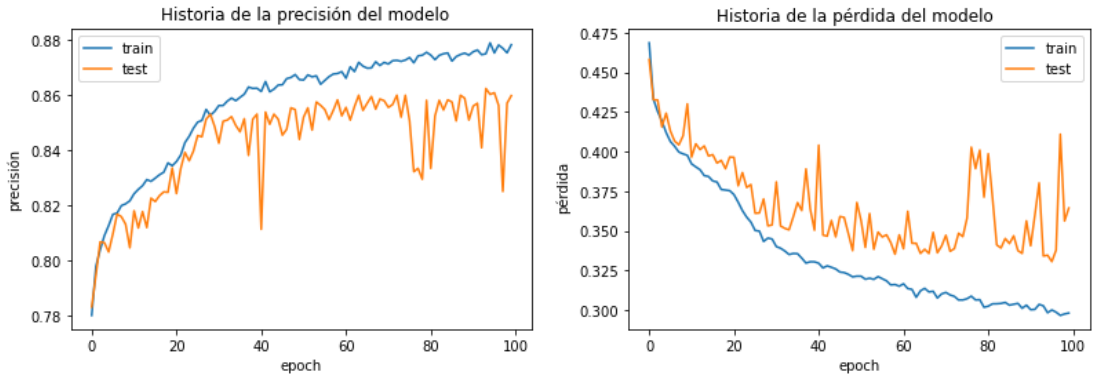


Ilustración 71.Gráficas de precisión y pérdida del modelo Todas las características 2C 100 – Ejecución 1 (Fuente propia)

Así, por un lado, se observa que **la evolución de la precisión del modelo no es correcta,** ya que **se frena su crecimiento** para los valores de validación o test una vez se superan las 30 épocas aproximadamente. Por otro lado, en la gráfica correspondiente a la evolución de la pérdida del modelo, se observa que **a partir de los 60 epoch aproximadamente se empieza a apreciar overfitting**, pues **los valores de pérdida de validación o test son altos y los de entrenamiento bajos.**

Por consiguiente, **para intentar minimizar este hecho**, se procede a ejecutar nuevamente el entrenamiento pero **esta vez bajando a 60 epoch** con el fin de intentar corregir el overfitting.

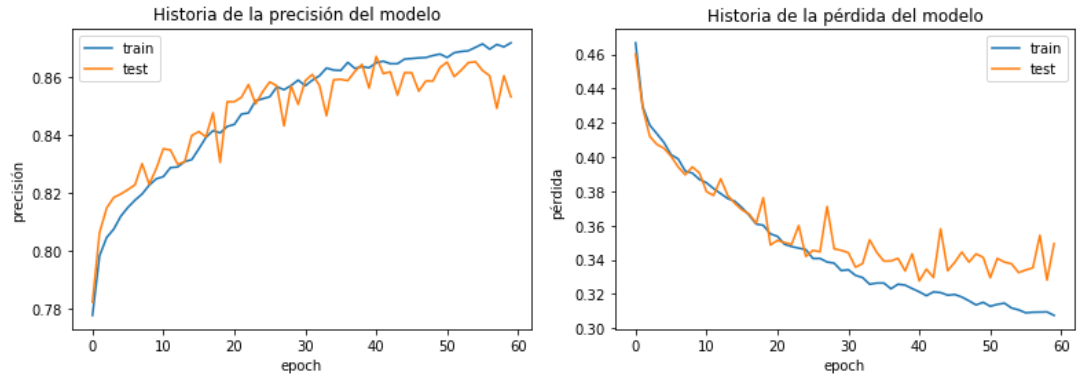


Ilustración 72. Gráficas de precisión y pérdida del modelo Todas las características 2C 100 – Ejecución 2 (Fuente propia)

De esta forma, tras este primer reajuste, la **precisión resultante disminuye a 85,7% con respecto a la inicial** y se observa nuevamente que **se frena su crecimiento** para los valores de validación o test. Además, **los valores de pérdida de validación o test empiezan a subir en la parte final del entrenamiento.** Así pues, para intentar reducir este **overfitting**, se procede a ejecutar nuevamente el entrenamiento, pero **esta vez bajando a 50 epoch** con el fin de minimizar este hecho.

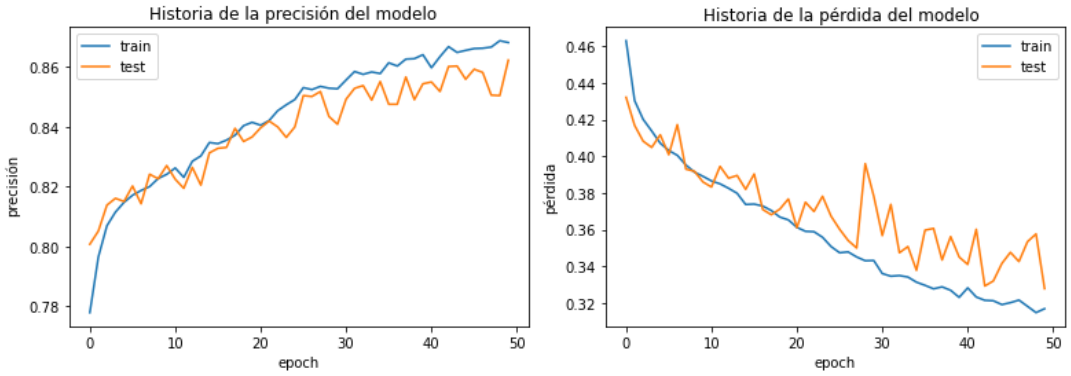


Ilustración 73. Gráficas de precisión y pérdida del modelo Todas las características 2C 100 – Ejecución 3 (Fuente propia)

Por consiguiente, tras el segundo reajuste, la **precisión resultante mejora a 86,6% con respecto a la ejecución anterior** y se aprecia que la tendencia de la misma ahora es correcta. Asimismo, se observa que **los valores de pérdida de validación o test presentan algunas subidas puntuales especialmente en la segunda mitad del entrenamiento**. Así pues, para intentar reducir este **comienzo de overfitting**, se procede a aumentar el conjunto de datos de aprendizaje pasando del 66,66% al 80%, con el fin de proporcionar al modelo más información para que mejore su entrenamiento.

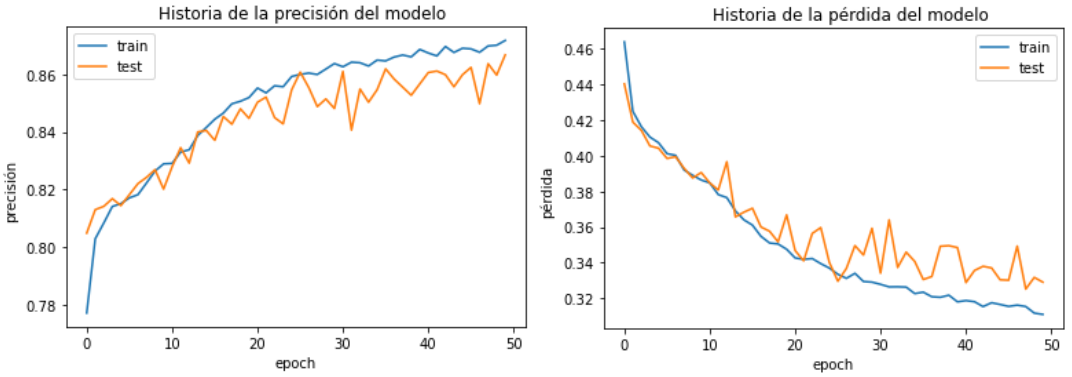


Ilustración 74. Gráficas de precisión y pérdida del modelo Todas las características 2C 100 – Ejecución 4 (Fuente propia)

Finalmente, tras el tercer reajuste, la **precisión resultante mejora a 87% con respecto a la ejecución anterior** y se observa nuevamente que la tendencia de la misma se mantiene correcta. Si se compara esta precisión con la inicial de 87,2% se observa que la pérdida de precisión es apenas perceptible ya que supone un 0,2%. Asimismo, se percibe que **los valores de pérdida de validación o test ya no presentan ninguna subida importante y van muy parejos con los valores de entrenamiento**, quedando por tanto el problema del **overfitting controlado** para este modelo.

9.5. Comparativa y conclusiones

En este apartado se procede a realizar la comparativa entre los 4 modelos seleccionados. Así, tras las diversas pruebas realizadas, se pretende escoger uno como finalista que será utilizado en el siguiente bloque para predecir los resultados de un conjunto de datos desconocido.

Una vez entrenados y reajustados los 4 modelos seleccionados, se procede a comparar las configuraciones finalmente aplicadas sobre los mismos, así como la precisión finalmente obtenida para cada uno de ellos. Además, para facilitar esta comparativa, se procede a asignar un identificador a cada uno de los modelos.

De esta forma, los reajustes aplicados que finalmente han mejorado a cada uno de los modelos resultan ser los siguientes:

Tabla 18. Comparativa de las configuraciones óptimas para los 4 modelos

Identificador	Modelo entrenado y reajustado	Configuración óptima de las probadas		Precisión obtenida
M1	RFE con árbol de decisión, normalización por lotes, 2 Capas y 100 epoch	**Epoch empleados**	**% de datos empleado en el entrenamiento**	86,7%
		80	80%	
M2	RFE con árbol de decisión, normalización por lotes, 2 Capas y 1000 epoch	**Epoch empleados**	**% de datos empleado en el entrenamiento**	87,8%
		180	80%	
M3	Todas las características, normalización por lotes, 1 capa y 500 epoch	**Epoch empleados**	**% de datos empleado en el entrenamiento**	85,8%
		500	66%	
M4	Todas las características, normalización por lotes, 2 capas y 100 epoch	**Epoch empleados**	**% de datos empleado en el entrenamiento**	87%
		50	80%	

Asimismo, la gráfica resultante comparando las distintas precisiones obtenidas para cada uno de los modelos presenta el siguiente aspecto:

Ilustración 75. Gráfica comparativa de la precisión obtenida en los 4 modelos (Fuente: propia)

Así, tal y como se puede apreciar en la gráfica aportada, en general los 4 modelos han conseguido precisiones muy similares entre sí, siendo M2 el modelo que mejor precisión ha logrado obtener alcanzando el 87,8% y M3 el que peor resultado ha obtenido con un 85,8%. Por consiguiente, para la elaboración del siguiente apartado, se decide finalmente escoger el modelo que ha presentado los mejores resultados de precisión, en este caso M2, mediante el cual se llevarán a cabo las predicciones de prueba.

10. Predicciones y resultados

En este apartado se procede a realizar predicciones utilizando para ello el modelo seleccionado en el apartado anterior, con el fin de analizar su eficacia mediante los resultados obtenidos.

Para la realización de las predicciones mediante el modelo M2 previamente escogido, se procede a elaborar de forma manual un **fichero con 100 cuentas de Twitter desconocidas** para el modelo. Así, se recopilan un total de **30 cuentas cuyo comportamiento coincide o da a pensar que son bots y 70 cuentas pertenecientes a personas reales**, junto con sus respectivas características extraídas de la API de Twitter mediante el mismo procedimiento que se empleó en el apartado *"7.4. Extracción de características de usuarios vía API".*

De esta forma, una vez formado el nuevo dataset, deberemos cargar dichos datos, tal y como puede observarse en "*Código 8. Carga de datos del fichero que contiene las características".* Asimismo, al igual que se hizo con el conjunto de entrenamiento y test, será necesario codificar aquellas variables que no sean numéricas, como se vio en "*Código 9. Codificación de variables para poder aplicar los métodos de selección*". Seguidamente, deberemos extraer las 7 características que recibirá como entrada el modelo (valor de X), así como su etiqueta de salida (valor de y), tal y como se detalló en *"Código 13. Método RFE para la extracción de las 10 mejores características con árbol de decisión"*

Tras ello, será necesario ejecutar la función "predict" sobre el modelo, que recibirá como parámetro de entrada el conjunto de características de los usuarios, el valor X, y tras ello devolverá un array con las predicciones que ha realizado con valores decimales comprendidos entre 0 y 1. Asimismo, para normalizar estos valores se redondearán y se realizará su conversión a tipo entero, dejando 0 y 1 como valores resultados.

```
from tensorflow.keras.models import load_model

#Se carga el modelo entrenado previamente
modelo = load_model('nombre_del_modelo_entrenado.h5')

#Se realiza la prediccion con los nuevos usuarios
predicciones = modelo.predict(X)

#redondeo a valores enteros de los resultados
predicciones_convertidas = predicciones.round().astype(int)
```

Código 27. Predicciones con el modelo entrenado y redondeo de resultados

Seguidamente, para almacenar estos resultados, se creará un fichero CSV donde se incluirá el identificador de cada cuenta, su etiqueta original de bot o humano, así como la etiqueta

calculada en la predicción, con el fin de poder comprobar así cuantos aciertos ha logrado alcanzar el modelo.

```
#Se forma el fichero final con los resultados
id_usuario = df['id']
tipo_usuario = df['tipo']
resultados = pd.concat([id_usuario, tipo_usuario,
           pd.DataFrame(data=predicciones_convertidas,
columns=['tipo_prediccion'])], axis=1)

resultados.to_csv('nombre_fichero_resultados.csv', index=False)
```

Código 28. Elaboración del fichero final con los resultados de las predicciones

De esta forma, una vez generado el fichero resultante de las predicciones tras comparar los resultados obtenidos con los esperados, se obtiene el siguiente balance:

Tabla 19. Resultados obtenidos de las predicciones

Tipo de cuenta	Total cuentas	Predicciones correctas	Predicciones erróneas
bot	30	20	10
human	70	61	9
Total	**100**	**81**	**19**

Asimismo, la gráfica correspondiente a los aciertos y fallos de las predicciones realizadas resulta ser la siguiente:

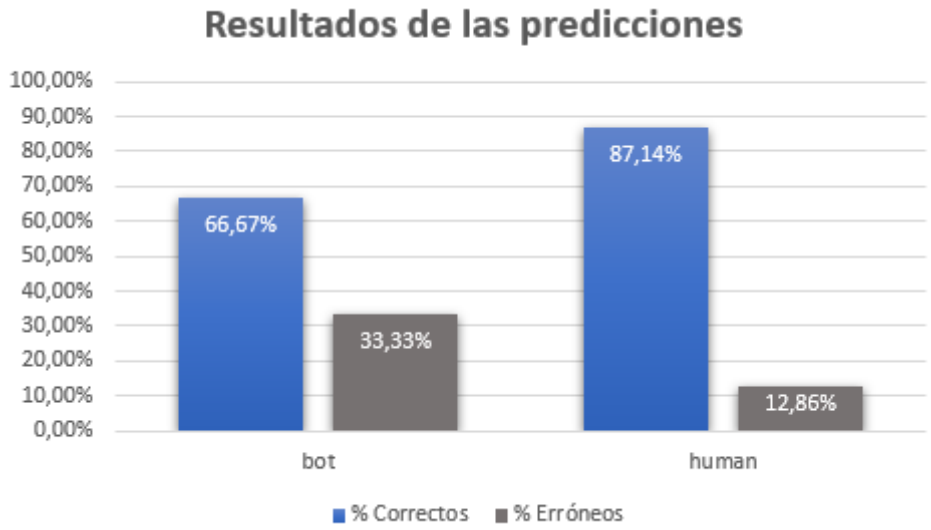


Ilustración 76. Gráfica comparativa con los resultados obtenidos para las predicciones (Fuente: propia)

Como se puede observar en los resultados recién mostrados, **el modelo entrenado funciona pues ha logrado clasificar de forma correcta al 81% de las cuentas, pero no es perfecto**. De esta

forma, si examinamos las estadísticas de aciertos y fallos por separado para bots y humanos, en primer lugar, se observa que en relación con los resultados obtenidos **para las cuentas que eran bots**, el modelo **ha acertado tan solo un 66,67% del total** de muestras, es decir, poco más de la mitad. Este hecho nos indica que el modelo no ha tenido el aprendizaje suficiente como para ser capaz de distinguir con mayor exactitud este tipo de cuentas, siendo probablemente una de las **principales causas** el **menor número de registros de bots con** respecto a los disponibles de humanos que han sido empleados **para llevar a cabo el entrenamiento** del modelo.

Por otro lado, se observa que en relación con los resultados obtenidos **para las cuentas que eran humanos**, el modelo **ha logrado clasificar de forma correcta al 87,14%** del total de las muestras, reflejando por tanto unos **resultados mucho más prometedores** y con una menor tasa de fallo.

Asimismo, **otra opción para mejorar los resultados obtenidos** sería **la estandarización o normalización de los datos de usuarios que se proporcionan como entrada al modelo**, pues tal y como se vio en las gráficas de precisión y pérdida de los entrenamientos de los modelos, existía mucho ruido a lo largo de estos entrenamientos debido a la **heterogeneidad de los datos que recibían como entrada**.

11. Conclusiones y trabajo futuro

Como conclusión principal sobre el proyecto desarrollado, se ha obtenido un modelo entrenado que detecta cuando una cuenta resulta ser un bot o un humano con una precisión alta. Sin embargo, este modelo no es perfecto. La detección de cuentas tipo bot le resulta difícil, mientras que la distinción de humanos la aborda con mayor eficacia.

Tras la extensa experimentación de configuración de modelos y su ejecución, se ha podido comprobar cómo el uso de más épocas (epoch) y/o de capas no implica una mejora de rendimiento y eficiencia en el entrenamiento del modelo, pues cada caso es particular y es necesario buscar y ajustar de forma progresiva la configuración que le va bien a cada modelo con el fin de evitar tanto el underfitting como el overfitting.

Otro dato que cabe destacar, resulta ser la gran cantidad de ruido presente en las gráficas de precisión y pérdida de cada uno de los modelos, pues se observan valores puntuales muy bajos de precisión y muy altos en la pérdida. Este hecho se entiende que es esperable debido a la heterogeneidad de los datos que recibe el modelo como entrada, pues habrá registros de usuarios que el modelo entiende de forma clara y otros que le resulten más raros o complejos de catalogar.

Asimismo, se ha visto que los métodos de selección de características probados en este proyecto funcionan de forma correcta, logrando que varios modelos configurados con un menor número de entradas, en este caso 7 características, alcanzasen precisiones muy similares e incluso mejores que aquellos configurados con 17 elementos de entrada. De esta forma, se puede confirmar que las características seleccionadas por estas técnicas resultan más relevantes y decisivas que otras a la hora de clasificar cada una de las cuentas como bot o humano.

El modelo seleccionado finalmente presenta varias opciones que podrían mejorar su eficacia y actuación en proyectos futuros, siendo algunas de estas propuestas de mejora las siguientes:

- Aumentar el conjunto general de datos empleado en el entrenamiento del modelo con el fin de ampliar así la cantidad de muestras disponibles para que la red neuronal disponga de un mayor abanico para distinguir con facilidad tanto bots como humanos.

- Ampliar de forma considerable el conjunto de datos pertenecientes a bots, pues tal y como se ha visto en los resultados obtenidos tras la realización de este proyecto, este tipo de cuentas cuesta más detectarlas. Por tanto, disponer de un abanico más amplio de ejemplos de cuentas de tipo bot, ayudará al modelo a que su entrenamiento sea más fructífero y logre así una mayor precisión a la hora de realizar predicciones.
- Normalizar o estandarizar los datos que recibe el modelo de entrada, es decir, las características de usuario, con el fin de eliminar de los mismos la información irrelevante. La baja calidad de algunos registros se ha visto que entorpecen de manera considerable el entrenamiento del modelo y generan mucho ruido, tal y como se pudo ver en las gráficas de pérdida y precisión, habiendo registros que el modelo entendía de forma correcta y otros que eran tan diferentes que no era capaz de clasificar correctamente.
- Incluir entre las características de entrada del modelo, datos calculados a partir de otros, como pueden ser algunos de los empleados en la propuesta de mi TFG como la media de retweets y tweets diarios, el ratio seguidores/seguidos, la media de publicaciones en base a la antigüedad de la cuenta, la periodicidad con la que los usuarios publican contenido o el resultado obtenido para cada cuenta en Botometer, entre otros.
- Hacer uso de la unidad de procesamiento de gráficos (GPU) para el entrenamiento de los modelos con TensorFlow en lugar de la CPU como se ha hecho en este proyecto, con el fin de acelerar todos los trabajos de cálculo, pues su paralelización permite una mayor velocidad de procesamiento.

Referencias

1. Wardle, C. (Investigador en la Universidad de Harvard) y Derakhshan, H. (Investigador en el centro Harvard Kennedy School's Shorenstein; Investigador en MIT Media Lab). *Information disorder: Toward an interdisciplinary framework for research and policy making.* 2017. Disponible en: http://tverezo.info/wp-content/uploads/2017/11/PREMS-162317-GBR-2018-Report-desinformation-A4-BAT.pdf
2. Zaracostas, J. (The lancet). *How to fight an infodemic.* 2020. Disponible en: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)30461-X/fulltext
3. OPS (Organización Panamericana de la Salud). *Entender la infodemia y la desinformación en la lucha contra la COVID-19.* Disponible en: https://www.paho.org/es/file/64245/download?token=vqDvd7jC
4. Rico Martínez, M. E. *Análisis de la credibilidad de cuentas en Twitter.* 2019. Disponible en: https://rua.ua.es/dspace/bitstream/10045/99807/1/Analisis_de_la_credibilidad_de_cuentas_en_Twitter_Rico_Martinez_Maria_Esther.pdf
5. Cerqueira, D. (ITS). *Introducción al concepto de bots.* 2020. Disponible en: https://itsrio.org/wp-content/uploads/2020/07/artigo1.pdf
6. Lohr, S. (Reportero sobre tecnología y economía en The New York Times). *It's True: False News Spreads Faster and Wider. And Humans Are to Blame.* 2018. Disponible en: https://www.nytimes.com/2018/03/08/technology/twitter-fake-news-research.html
7. BBC. *WhatsApp restricts message-sharing to fight fake news.* 2019. Disponible en: https://www.bbc.com/news/technology-46945642
8. Dwoskin, E. (Corresponsal de Silicon Valley en The Washington Post). *Facebook is rating the trustworththiness of its users on a scale from zero to 1*. 2018. Disponible en: https://www.washingtonpost.com/technology/2018/08/21/facebook-is-rating-trustworthiness-its-users-scale-zero-one/
9. BBC. *Facebook employs UK fact-checkers to combat fake news.* 2019. Disponible en: https://www.bbc.com/news/technology-46836897
10. Sharma, S. (Profesor asistente en el colegio BSSS Bhopal, Madhya Pradesh – India) y Kumar Sharma, V. (Profesor asistente en OIST Bhopal, Madhya Pradesh – India). *Cyber Crime análisis on Social Media.* 2020. Disponible en: https://bssspublications.com/Home/IssueDetailPage?IsNo=258

11. P, Sowmya (Universidad de Mumbai – Faculta de Ingeniería de Pillai) y Chatterjee, Madhumita (Universidad de Mumbai – Faculta de Ingeniería de Pillai). *Detection of Fake and Cloned Profiles in Online Social Networks. Proceedings 2019: Conference on Technologies for Future Cities (CTFC).* 2019. Disponible en: https://ssrn.com/abstract=3349673
12. P, Sowmya (Universidad de Mumbai – Facultad de Ingeniería de Pillai) y Chatterjee, Madhumita (Universidad de Mumbai – Facultad de Ingeniería de Pillai). *Detection of Fake and Clone accounts in Twitter using Classification and Distance Measure Algorithms.* 2020. Disponible en: https://ieeexplore.ieee.org/abstract/document/9182353
13. Sahoo, S.R. (Profesor asistente en el Instituto de Tecnología Vellore) y Gupta, B.B. (Departamento de Ingeniería Informática en el Instituto Nacional de Tecnología, Kurukshetra, Haryana, India). *Real-Time Detection of Fake Account in Twitter Using Machine-Learning Approach.* 2021. Disponible en: https://doi.org/10.1007/978-981-15-1275-9_13
14. Beth, T., Borcherding, M. y Klein, B. *Valuation of trust in open* networks. Gollmann D. (eds) Computer Security — ESORICS 94. ESORICS 1994. Disponible en: https://doi.org/10.1007/3-540-58618-0_53
15. K. Ojo, A. (Departamento de ciencias de la computación, Universidad de Ibadan, Nigeria). *Improved Model for Detecting Fake Profiles in Online Social Network: A Case Study of Twitter.* Journal of Advances in Mathematics and Computer Science, 33(4), 1-17. 2019. Disponible en: https://doi.org/10.9734/jamcs/2019/v33i430187
16. Kaubiyal, J. (Instituto Nacional de Tecnología, Kurukshetra, Thanesar, Haryana, India) y Jain, AK. (Instituto Nacional de Tecnología, Kurukshetra, Thanesar, Haryana, India). *A Feature Based Approach to Detect Fake Profiles in Twitter.* 2019. Disponible en: https://doi.org/10.1145/3361758.3361784
17. Jabardi, M. (Universidad de Kufa) y Hadi, AS. (Universidad de Babilonia). *Twitter Fake Account Detection and Classification using Ontological Engineering and Semantic Web Rule Language.* Karbala International Journal of Modern Science: Vol. 6: Iss. 4, Article 8. 2020. Disponible en: https://doi.org/10.33640/2405-609X.2285
18. BalaAnand, M. (Departamento de Ingeniería y Ciencias de la Computación, V.R.S. Facultad de Ingeniería y Tecnología, Viluppuram, India), Karthikeyan, N. (Departamento de MCA, Facultad de Ingeniería SNS, Coimbatore, India), Karthik, S. (Departamento de Ingeniería y Ciencias de la Computación, Facultad de Tecnología de SNS, Coimbatore, India) y otros. *An enhanced graph-based semi-supervised learning algorithm to detect*

fake users on Twitter. J Supercomput 75, 6085–6105. 2019. Disponible en: https://doi.org/10.1007/s11227-019-02948-w

19. Documentación oficial sobre el sistema operativo Ubuntu con información para desarrolladores. Disponible en: https://ubuntu.com/desktop/developers
20. Documentación sobre cómo instalar Python 3.8 en Ubuntu. Disponible en: https://linuxize.com/post/how-to-install-python-3-8-on-ubuntu-18-04/
21. Prasad Acharya, D. (Escritor de contenido tecnológico en Geekflare). *Los 10 mejores IDE de Python para potenciar el desarrollo y la depuración.* 2020. Disponible en: https://geekflare.com/es/best-python-ide/
22. Documentación oficial de PyCharm y portal de descarga. Disponible en: https://www.jetbrains.com/pycharm/
23. Documentación oficial de la librería Tweepy sobre el uso de la API de Twitter. Disponible en: https://docs.tweepy.org/en/latest/api.html
24. AprendeIA. (Portal web destinado al aprendizaje y la oferta de recursos para aprender Machine Learning). *Introducción a la librería Scikit-Learn de Python. 2018.* Disponible en: https://aprendeia.com/libreria-scikit-learn-de-python/
25. Portal oficial de la librería TensorFlow. Disponible en: https://www.tensorflow.org/
26. Cutipa, G. (Profesional IT con experiencia en la administración de red con Linux y otras tecnologías). *Cómo instalar Pentaho Data Integration (PDI) Tool en Ubuntu.* 2021. Disponible en: https://guidocutipa.blog.bo/como-instalar-pentaho-data-integration-pdi-tool-en-ubuntu/
27. Portal oficial de descarga de la herramienta Pentaho Data Integration. Disponible en: https://sourceforge.net/projects/pentaho/files/latest/download
28. Portal oficial del proyecto Jupyter. Disponible en: https://jupyter.org/
29. Portal oficial de descarga de la distribución Anaconda en su edición individual. Disponible en: https://www.anaconda.com/products/individual/
30. S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi. *The Paradigm-Shift of Social Spambots: Evidence, Theories, and Tools for the Arms Race*. WWW '17 Proceedings of the 26th International Conference on World Wide Web Companion, 963-972. 2017. Disponible en: https://dl.acm.org/doi/10.1145/3041021.3055135
31. S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi. *Fame for sale: efficient detection of fake Twitter followers.* arXiv:1509.04098 09/2015. Elsevier Decision Support Systems, Volume 80, December 2015, Pages 56–71. 2015. Disponible en: https://arxiv.org/abs/1509.04098

32. Martín Gutiérrez, D. (Científico de datos y contribuidor de la plataforma Kaggle). Dataset publicado "*Twitter Bots Account – An updated for Twitter Bot account Detection".* Disponible en: https://www.kaggle.com/davidmartngutirrez/twitter-bots-accounts
33. Jain, C. (Contribuidora de la plataforma Kaggle). Dataset publicado *"detecting twitter bot data".* Disponible en: https://www.kaggle.com/charvijain27/detecting-twitter-bot-data
34. Limão, J.P. (Contribuidor de la plataforma Kaggle). Dataset publicado *"Bots in Twitter".* Disponible en: https://www.kaggle.com/joopedrolimo/bots-in-twitter
35. Shinde, S.V. (Científico de datos y contribuidor de la plataforma Kaggle). Dataset publicado *"Tweeter bots Dataset".* Disponible en: https://www.kaggle.com/saurabh778/tweeter-bots-dataset
36. Brownlee, J. (Desarrollador profesional). *How to Choose a Feature Selection Method For Machine Learning.* 2019. Disponible en: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/
37. Martinez Heras, J. (Científico de datos senior con amplia experiencia en el campo de la Inteligencia artificia). *¿Clasificación o Regresión?* 2020. Disponible en: https://www.iartificial.net/clasificacion-o-regresion/
38. Brownlee, J. (Desarrollador profesional). *Recursive Feature Elimination (RFE) for Feature Selection in Python.* 2020. Disponible en: https://machinelearningmastery.com/rfe-feature-selection-in-python/
39. Brownlee, J. (Desarrollador profesional). *TensoFlow 2 Tutorial: Get Started in Deep Learning with tf.keras.* 2020. Disponible en: https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/
40. *Nautiyal, D. (Interno en GeeksforGeeks). Underfitting and Overfitting.* 2021. Disponible en: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/