UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

"SenseQ: Bot que responde a preguntas a través de redes sociales"

AUTOR: Carlos Menéndez Martínez

DIRECTOR: Cristian González García

Resumen

Este proyecto se centra en la creación de una cuenta de Twitter automatizada o *bot* capaz de responder a las preguntas que le planteen los usuarios de la red social. Para ello, se mencionará a la cuenta con la cuestión y esta se encargará de analizar el tipo de pregunta y dar la respuesta más acorde posible.

A nivel técnico se requiere crear una cuenta automatizada que sea capaz de escribir *tweet*s, sería en este caso el equivalente al *front end*, un modelo clasificador de lenguajes que sea capaz de analizar el texto enviado por el usuario, clasificarlo en distintas categorías y un sistema capaz buscar la respuesta utilizando diferentes APIs y servicios externos.

La idea es facilitar las posibles modificaciones futuras en la mayor medida posible, permitiendo añadir nuevas APIs o nuevas formas de procesar/clasificar el texto sin tener que realizar modificaciones en el resto del sistema.

Palabras Clave

Procesamiento de Lenguaje Natural, Machine Learning, API, Python, Twitter, Preguntas, Bot.

Abstract

This project focuses on the creation of an automated Twitter account or *bot* capable of answering the questions given by the social network users. To do that, they will have to mention said account and it will be responsible for analyzing what kind of question is and give an answer accordingly.

On a technical level, an automated twitter account capable of writing tweets is required, which in this case it will act as the front end of the system. Also, a classifying model that can analyze text sent by users and sort it into different categories and a system able to search the answer through different APIs and external services.

The idea is to make future modifications of the system as easy as possible, allowing new APIs or new ways of processing/sorting the text to be added without having to procure changes in the rest of the system.

Keywords

Natural Language Processing, Machine Learning, API, Python, Twitter, Questions, Bot.

Índice General

Índice de Figuras

Índice de tablas

Índice de Imágenes

Capítulo 1. Memoria del Proyecto

Este capítulo resume brevemente la motivación, los objetivos y alcance del proyecto.

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

En este proyecto se realizó un sistema capaz de dar una respuesta a las preguntas que realizan los usuarios de Twitter, con la intención de ofrecer una alternativa dentro de la red social a otros sistemas que realizan una función similar.

Para conseguir desarrollar este sistema se establecieron como objetivos del proyecto la realización de un sistema capaz de comunicarse con Twitter con el fin de detectar el momento en el que un usuario le ha mencionado realizando una consulta, para que el resto del sistema la procese y obtenga una respuesta, consiguiendo así, responderla adecuadamente.

Así mismo, debe poder comunicarse con los servicios que se han precisado necesarios para la búsqueda de respuestas, como son la API de Wikipedia o Google.

Finalmente, el objetivo fundamental de este proyecto consistió en el aprendizaje y creación de varios modelos clasificadores de texto mediante técnicas de *machine learning*, así como el estudio de los resultados que nos ofrecen y las modificaciones que se hayan podido realizar hasta encontrar el modelo adecuado.

Con todo, este proyecto se llevó a cabo utilizando Python como único lenguaje, y para lo referente a la clasificación de textos las bibliotecas de NLTK y Scikit-Learn.

1.2 Resumen de Todos los Aspectos

El proyecto se compondrá de los siguientes apartados:

- **Capítulo 1.** Resumen de la motivación, objetivos y el alcance al que se quiere llegar con el proyecto.
- **Capítulo 2.** Introducción del proyecto, justificación, objetivos y estudio de alternativas
- **Capítulo 3.** Aspectos teóricos relevantes para el proyecto.
- **Capítulo 4.** Planificación del proyecto y presupuesto inicial.
- **Capítulo 5.** Evaluación y comparativa de los resultados obtenidos durante el desarrollo del proyecto.
- **Capítulo 6.** Análisis del sistema. Definiremos todos los aspectos relativos al análisis, requisitos, especificación del plan de pruebas...
- **Capítulo 7.** Diseño del sistema. Definiremos la arquitectura del sistema, el diseño de sus clases, los distintos diagramas que lo definen y especificaremos de forma técnica el plan de pruebas.
- **Capítulo 8.** Implementación del sistema, hablaremos de los estilos lenguajes, herramientas y paquetes utilizados, así como de los problemas que se han encontrado y de una descripción de forma detallada de cada clase.
- **Capítulo 9.** Desarrollo de las pruebas, veremos los resultados del plan de pruebas.
- **Capítulo 10.** Manuales para la instalación y ejecución del sistema, así como para el programador.
- **Capítulo 11.** Conclusiones y posibles ampliaciones futuras.
- **Capítulo 12.** Planificación y presupuesto finales.
- **Capítulo 13.** Referencias bibliográficas.
- **Capítulo 14.** Apéndices.

Capítulo 2. Introducción

En este capítulo se realizará una breve justificación del proyecto y se expondrá la situación actual en la que se encuentran los proyectos similares.

2.1 Justificación del Proyecto

En la actualidad, la población cada vez más se ve sobrecargada de la inmensa cantidad de información que tiene a su alcance. Tener a nuestra disposición tal cantidad de conocimiento, el cual además no cesa en aumentar día a día, tiene por supuesto innumerables ventajas: aumenta a la culturización de la población, beneficia el desarrollo del sistema educativo, permite que nos especialicemos en ámbitos y temas de interés, etc. Sin embargo, también tiene desventajas. A mi parecer, uno de los principales inconvenientes es que en algunos casos la búsqueda de datos concretos resulta a veces infructuosa, en el sentido de que obtenemos más información de la que pretendemos, siendo parte de esta información compleja, y no siempre deseamos ampliarla.

A pesar de ser consciente de la existencia en el mercado de otras alternativas adecuadas para obtener respuestas cortas, concisas y sencillas a preguntas concretas, como podrían ser los asistentes virtuales, acerca de los cuales hay que mencionar que no es su única función ni utilidad, consideramos que la población también podría beneficiarse de la existencia y utilización de un sistema exclusivo de respuestas a preguntas cortas y específicas relacionadas con un tema concreto, que además sea acceso libre y de fácil utilización.

Con esto en mente se propone desarrollar un sistema que pueda responder preguntas cuando se esté en una red social, en este caso se eligió Twitter, debido a las facilidades que esta ofrece de cara a crear una cuenta automatizada y ser de las redes sociales con una política más abierta de cara a la extracción de datos de la plataforma, además de ser de las que más se adapta en formato a la idea del proyecto.

Cualquier usuario registrado debe poder enviarle un *tweet* con su pregunta al bot y este responderle a este mismo *tweet*.

Para la realización de este proyecto se utilizarán técnicas tanto de procesamiento de lenguajes naturales (NLP) como de *machine learning*, con la idea de poder comprender que se están preguntando los usuarios y desarrollar un sistema capaz de clasificar cada pregunta en función de su temática, una vez realizada su clasificación el objetivo será utilizar diferentes servicios para encontrar respuestas “oficiales” a dichas preguntas, citando la fuente donde se ha encontrado la información.

2.2 Objetivos del Proyecto

El objetivo principal es aprender a realizar desde cero un proyecto de estas características, diseñando y probando cada capa que lo compone.

Más específicamente, el proyecto se puede dividir en los siguientes objetivos:

1. Estudiar la situación actual de alternativas a nuestro sistema, tanto en la red social como fuera de ella.
2. Crear una cuenta de Twitter automatizada para recoger el texto de un *tweet* de un usuario y responder al mismo.
3. Evaluar distintos modelos de *machine learning*, según sus algoritmos y configuraciones, para encontrar el más adecuado a nuestro proyecto.
4. Normalizar un texto extraído de un *tweet*, siguiendo siempre los mismos pasos de preprocesamiento como pueden ser eliminación de palabras vacías, lematización, *tokenización*…
5. Categorizar el texto de una pregunta según su tipo (salud, economía, deportes…), para poder buscar la respuesta más adecuada a este, gracias al modelo entrenado mediante *machine learning* previamente.
6. Conectarse a distintas APIs, especializadas en distintas categorías.
7. Utilizar estas APIs para buscar una respuesta según la pregunta del usuario.

2.3 Estudio de la Situación Actual

Tras una búsqueda exhaustiva de la situación actual relativa a nuestro proyecto, en este apartado se evaluarán las alternativas que se encuentran a este proyecto.

2.3.1 Evaluación de Alternativas

Como alternativas similares a nuestro proyecto, tan sólo se ha encontrado un bot en Twitter que pueda asemejarse al nuestro, el cual se explicará más adelante.

Sin embargo, es importante mencionar que una alternativa obvia, que resulta accesible a casi cualquier persona y que también cumple la función que se pretende conseguir con este sistema son los asistentes virtuales.

En cuanto a lenguajes en los que realizar el proyecto, se evaluarán dos alternativas encontradas, Java y Ruby.

2.3.1.1 Asistentes virtuales

Los asistentes virtuales se han convertido en una parte del día a día de muchas personas en la actualidad, con muchas alternativas disponibles en función del dispositivo que utilicemos. Los más comunes son el Asistente de Google, Siri (para dispositivos de Apple) o Alexa (para dispositivos de Amazon).

Estos asistentes funcionan principalmente por voz, y son capaces de realizar una función similar a la de nuestro proyecto, además de otras funcionalidades que satisfacen otras necesidades.

Como muestra del alcance que llegan a obtener estos asistentes, el estudio Voice Consumer Index [2] de la empresa Vixen realizado en EE. UU., Reino Unido y Alemania, determinó que aproximadamente el 60% de los entrevistados han utilizado uno de estos asistentes, con más de un 30% de personas que reconocen utilizarlo cada día.

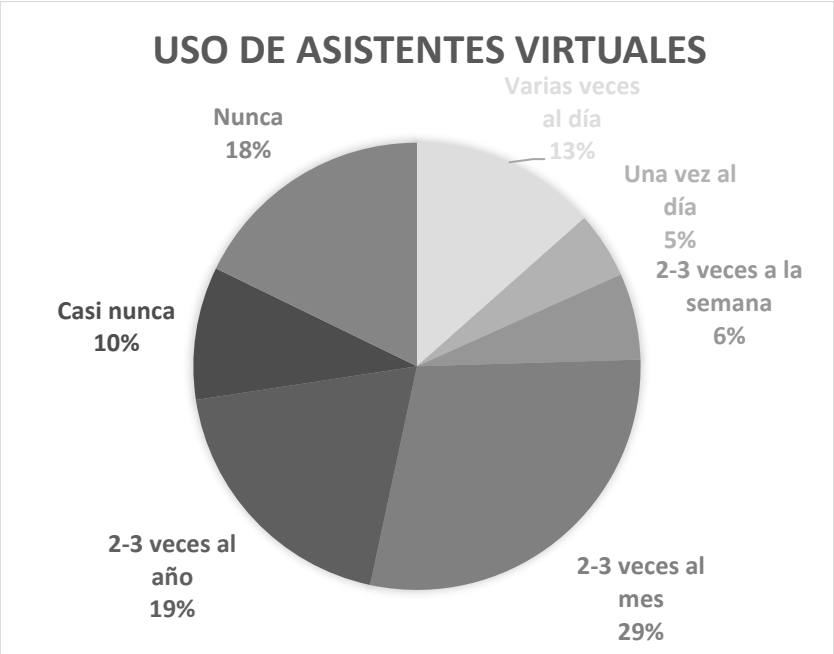

Imagen 1. Uso de asistentes virtuales según el VCI.

En el mismo estudio también se preguntó por el uso que se da a estos asistentes, lo que puede ayudar de cara a saber que cuestiones serán más habituales entre los usuarios.

Categoría	Uso
Tiempo	0,75
Música	0,71
Noticias	0,64
Entretenimiento	0,62
Compras	0,54
Comida	0,52
Salud	0,51
Paquetes y envíos	0,49
Servicios	0,49
Reservas	0,47
Deportes	0,46
Moda	0,45
Viaje	0,43

Tabla 1. Uso de los asistentes virtuales según el tipo.

2.3.1.2 AnswerBot

Imagen 2. Logo de AnswerBot.

AnswerBot [3] es un bot de Twitter que por su descripción es similar a nuestro proyecto. Está realizado con el software de Microsoft QnAMaker [4], una herramienta que facilita la creación de bots conversacionales sin necesidad de tener experiencia en programación y Twidge, un cliente de terminal para Twitter [5].

El proyecto está descontinuado desde 2019 y viendo los *tweet*s publicados, parece que nunca llegó a funcionar según su descripción, llegando sólo a responder respuestas predeterminadas.

2.3.1.3 ChatGPT

Imagen 3. Logo ChatGPT.

ChatGPT [6], [7] es un chatbot de inteligencia artificial desarrollado en 2022 por OpenAI especializado en el diálogo. Consiste en un gran modelo de lenguaje entrenado con técnicas de *machine learning* de aprendizaje supervisado y de refuerzo (ver *3.2Machine Learning*).

Este tipo de proyectos de inteligencia artificial son de gran notoriedad en la actualidad, debido a los avances que se han realizado en el campo de la inteligencia artificial, así como el crecimiento en el interés que se tiene en proyectos de este tipo, no sólo por motivos de entretenimiento si no también económicos.

En cuanto a las diferencias con el sistema propuesto, ChatGPT emula la forma de escribir humana y mantiene una conversación con el usuario, generando las respuestas mediante modelos de *machine learning*. Nuestro sistema, en cambio, busca responder de forma directa al usuario. También utiliza modelos de machine learning, pero estos son para la clasificación de las preguntas ya que las respuestas se obtienen mediante la integración de APIs.

2.3.1.4 Lenguaje

Con respecto a los lenguajes se han evaluado dos alternativas:

2.3.1.4.1 Java

Imagen 4. Logo de Java.

Java es un lenguaje orientado a objetos de propósito general diseñado para tener el mínimo posible de dependencias. Creado en 1996 por Sun Microsystems (Oracle) [5]. Sigue el paradigma de la orientación a objetos y su sintaxis se inspira principalmente en C y C++. Es un lenguaje multiplataforma, capaz de ejecutarse en la mayoría de los sistemas operativos. Se utiliza para el desarrollo de todo tipo de aplicaciones, móviles, en la nube, chatbots o aplicaciones empresariales [8]. En términos de popular Java tan sólo se encuentra por detrás de Python según el índice PYPL [9] y, en cuarto lugar, adelantándole C y C ++ según el ranking de Tiobe [10].

2.3.1.4.1.1 Ventajas

- Estabilidad, pero requiere más código.
- Velocidad de ejecución.
- Robustez, Java hace un énfasis especial en ayudar a los usuarios a no tener errores.
- Twitter ofrece una API oficial para desarrollar en ella.

2.3.1.4.1.1.1 Inconvenientes

- Para proyectos de *machine learning*, requiere de más código que Python para poder funcionar.
- No puede compilar si hay cualquier tipo de error o bug en el código, lo que dificulta el desarrollo[11].

2.3.1.4.2 Ruby

Imagen 5. Logo de Ruby.

Ruby es un lenguaje de programación open-source de orientación a objetos pero que también permite programación funcional. Creado en 1993 por Yukihiro Matsumoto, inspirándose en Python y Perl [10], [12]. Ruby se encuentra actualmente en la posición 17 según el ranking de Tiobe [10], con un 0,80% de popularidad. Y según el ranking de PYPL en la posición 15 con un 1,13% de popularidad [9].

2.3.1.4.2.1 Ventajas

- Soporte en la API de Twitter, con ejemplos de uso con código.
- Muy simple

2.3.1.4.2.2 Inconvenientes

- No está tan extendido como Python o Java, por lo que es más complicado encontrar información.
- Menor cantidad de librerías con respecto a Python tanto de *machine learning* como generales.
- El tiempo de ejecución de los programas escritos en Ruby tiende a ser más lento que el de sus competidores.
- Tanto el lenguaje como su documentación está centrado sobre todo en el desarrollo web [13].

Capítulo 3. Aspectos Teóricos

En este capítulo se explorará la teoría referente a nuestro proyecto.

3.1 Procesamiento de Lenguaje Natural

El procesamiento de lenguajes naturales es el campo de la Inteligencia artificial que se ocupa de la investigación en la manera de comunicar máquinas y personas mediante el uso de lenguas ordinarias [14].

Si bien se podría hacer para cualquier Lengua existente, en la realidad, solo las más habladas o utilizadas tienen aplicaciones en uso, por motivos tanto económicos como prácticos [15].

3.2 Machine Learning

Machine learning es una rama del campo de la Inteligencia Artificial [16], en la cual, a través de distintos algoritmos, se provee a las computadoras con la capacidad de identificar patrones en grandes cantidades de datos y elaborar predicciones con ellos. Este aprendizaje permite a los sistemas realizar tareas específicas de forma autónoma sin necesidad de programación [17].

Existen tres categorías en las que se puede dividir el aprendizaje en este campo [7], [18]:

- Supervisado: el aprendizaje se realiza gracias a etiquetas previamente asociadas a los datos.
- No supervisado: los datos que se proveen no están etiquetados, es el sistema el que debe encontrar la forma de encontrar patrones que le permita organizarlos.
- Por refuerzo: aprende a partir de la propia experiencia, decide cuál es la mejor decisión mediante un proceso de prueba y error.

3.3 ChatBot

Un chatbot, o bot conversacional, es un software automatizado que se encarga de interactuar con un usuario mediante el lenguaje en una plataforma [19].

Estos chatbots varían en complejidad y capacidad de comprensión en función de su uso, en el ámbito empresarial se usa para dar al usuario información específica de la empresa cuando este la solicita, dando una información muy definida previamente y sin dar respuesta a preguntas que salgan de su dominio.

En la actualidad, existen muchos ejemplos que ya se han instaurado en la vida cotidiana de la gente como son por ejemplo Siri, Alexa, Cortana el Asistente de Google. Estos son enmarcados como asistentes inteligentes ya que son capaces de generar un comportamiento inteligente con el usuario, no teniendo respuestas predefinidas y adaptándose a las peticiones del usuario.

Recientemente, las inteligencias artificiales aplicadas a los chatbots se han vuelto muy populares, como es el caso de ChatGPT:

3.4 Preprocesamiento de texto

El preprocesamiento es el primer paso en NLP, en él se prepara el texto para la construcción del modelo, suele componerse de diferentes etapas, si bien es a disposición de cada uno cuanto preprocesar el texto.

3.4.1 Lematización

La lematización es un proceso por el cual se intenta encontrar el lema o raíz de esta, con esto se pretende evitar ambigüedades y estandarizar las palabras del texto.

3.4.2 *Stopwords*

Las *stopwords* o palabras vacías, son las palabras que no aportan significado a un texto, si bien a nosotros nos sirven para dar coherencia y cohesión a nuestro lenguaje, son principalmente las conjunciones, artículos, preposiciones y adverbios. Eliminar estas palabras aligera enormemente la carga de trabajo que nuestro procesador tendrá que realizar.

3.5 Algoritmos *Machine Learning*

Para la clasificación del texto se probaron diferentes algoritmos, se explicará ahora el funcionamiento de cada uno.

3.5.1 Naive Bayes

Un clasificador de Naive Bayes funciona sacando una serie de características de para cada categoría y asumiendo que son independientes unas de otras. Es llamado así por el teorema de Bayes y por la palabra en inglés *naive* que significa ingenuo [20].

El ejemplo más habitual para entenderlo es la clasificación de mensajes en spam o no spam, el algoritmo buscará las palabras más comunes para cada tipo de mensaje, por ejemplo, la palabra dinero para los mensajes de tipo spam será más recurrente que para los que no son spam, y guardará la frecuencia con la que aparecen cada una. Una vez entrenado, cuando se quiere hacer una predicción el algoritmo buscará para cada palabra, con qué frecuencia aparecen tanto en los mensajes de spam como no spam, y hará una clasificación acorde. Así pues, un mensaje con varias menciones al dinero será más probable que sea clasificado como spam que como no spam [21].

3.5.2 Decision Tree

Un árbol de decisiones es un clasificador que es capaz de predecir la categoría de nuestro objetivo mediante reglas de decisión simples. Para realizar la predicción se empieza por la raíz del árbol y se va comparando con los atributos correspondientes, siguiendo las distintas ramas hasta llegar al final del árbol, donde se obtiene la categoría presumida [22].

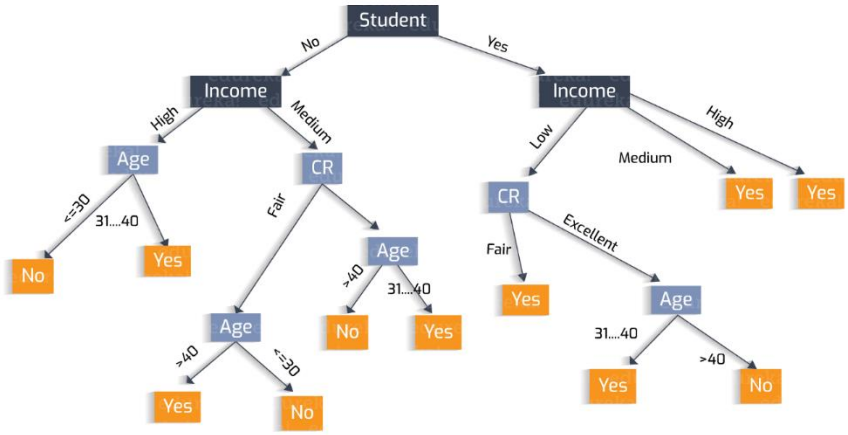

Imagen 6. Decision Tree. Fuente: Heartbeat.

Para ejemplificarlo, consideremos el supuesto de que tenemos una lista de personas con ciertos datos personales, su edad, su ocupación, su género, algún interés. Y queremos saber, por ejemplo, si es aficionado a los deportes de motor. El algoritmo creará un árbol de decisiones en función de los datos de la lista, y creará un árbol de decisiones acorde, si por ejemplo considera que las personas mayores de 20 años son más propensas a ser aficionados, cuando recorriendo el árbol se llegue a esa decisión, se dará una mayor probabilidad de que sean a los mayores de 40.

Una vez creado el árbol cuando se introducen los datos de una persona nueva para que se haga la predicción, recorrerá uno a uno cada condición que ha creado, lo cual alterará negativa o positivamente la probabilidad de que esa persona cumpla lo que buscamos.

3.5.3 Random Forest

Un Random Forest es un conjunto de Decision Trees, cuando se quiere hacer una clasificación. Cada uno de ellos predice un resultado, y se elige el que más apariciones tenga en el conjunto [23].

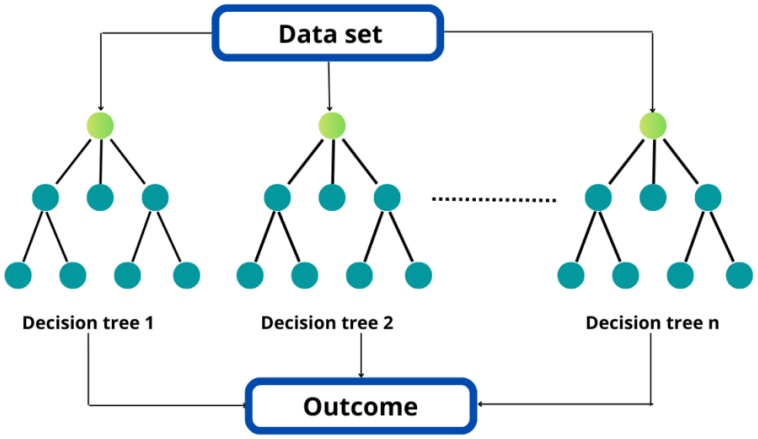

Imagen 7. Random Forest. Fuente: CopyAssignment.

Siguiendo con el ejemplo anterior del Decision Tree, Random Forest, realizaría los mismos pasos, pero en lugar de construir un solo árbol de decisiones, construye tantos como el usuario le indique (100 es el valor por defecto en scikit-learn), así obtendríamos un grupo de árboles, pongamos 100, capaz de clasificar si a una persona le gustan o no los deportes de motor. Cuando el usuario introduce unos valores para realizar una nueva predicción, todos ellos realizan su clasificación, y se observa cual es el resultado más común, si de 100 árboles, 80 han decidido que el usuario es aficionado al motorismo, esa será la decisión del clasificador.

3.5.4 K-Nearest Neighbors

K-Nearest Neighbors (KNN) asume similitudes entre los datos nuevos y los que se le han dado previamente, clasificando lo nuevo en la categoría más similar [18].

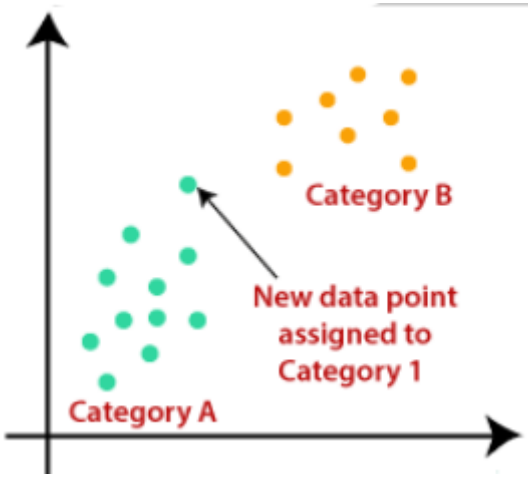

Imagen 8. Ejemplo KNN. Fuente: JavaPoint.com.

Si tenemos una lista de imágenes etiquetadas como gatos o perros, el algoritmo extraerá las características más comunes. A la hora de realizar una predicción fuera de los datos proporcionados previamente, se buscarán las características de la imagen que se introduce, y se clasificará buscando con cuál de las dos tiene más similitudes.

3.5.5 SVM

En un algoritmo SVM (*Support Vector Machine* o en español máquina de vectores de soporte) trata de construir un plano de varias dimensiones basándose en los datos ofrecidos en el aprendizaje, con el fin de poder separar las distintas clases de los datos [25].

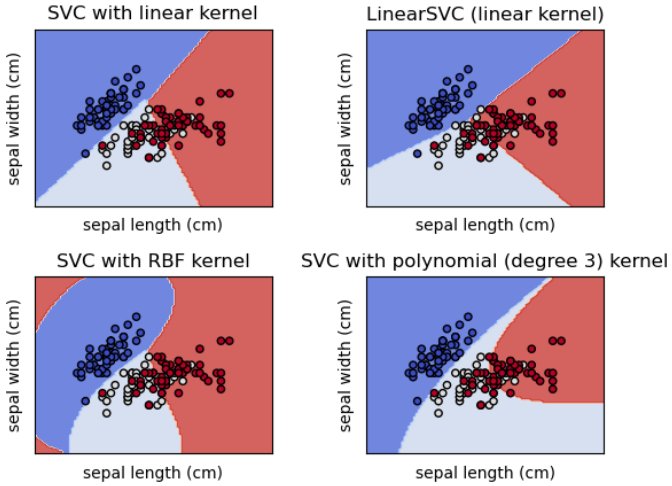

Imagen 9. Ejemplos de planos creados por SVM. Fuente: Scikit-Learn.

Por ejemplo, imaginemos que, como anteriormente queremos clasificar imágenes según si son perros, gatos o pájaros, nuestro algoritmo crearía tres planos de forma similar a la imagen superior. Después cuando quiere realizar una predicción situaría los datos en el plano que le corresponda, y ese sería nuestra predicción.

3.5.6 Métricas

Para poder saber la precisión y calidad que ofrecen los anteriores algoritmos es necesario tener una forma de compararlos [26], a continuación, se definen los términos utilizados:

Se define ***precision,*** o en español precisión, como la relación entre los positivos acertados y el total de positivos (acertados y fallados). Se calcula siguiendo la ecuación:

$$\frac{TP}{TP + FP}$$

Se entiende ***accuracy***, o exactitud, como el total de aciertos (positivos y negativos) entre el total de predicciones. La fórmula para su obtención es:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

El ***recall,*** o exhaustividad, es la relación entre los positivos acertados y la suma de éstos con los falsos negativos. En forma de ecuación será:

$$\frac{TP}{TP + FN}$$

Por último, ***F1-score*** (valor F1) es una forma de reunir la precisión y el *Recall*, concretamente, la media armónica de estos dos. Es un valor que puede ir de 0 a 1 siendo 1 el resultado óptimo [27].Sigue la siguiente fórmula:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Capítulo 4. Planificación del Proyecto y Presupuesto Iniciales

En este apartado se realizará la planificación y el presupuesto iniciales de nuestro proyecto.

4.1 Planificación Inicial

Primero se estudiará la planificación. En ella se establecen los roles que conforman el proyecto, siendo todos ellos realizados por el autor, así como las tareas a llevar a cabo y su duración. Los recursos del proyecto serán los siguientes:

- Jefe de Proyecto: determina los objetivos del proyecto, crea el plan a seguir y coordina a los demás recursos.
- Arquitecto de Software: diseña el sistema, elige su arquitectura y crea los estándares a seguir.
- Analista: realiza las funciones de investigación, generación de requisitos y documentación
- Programador Senior: se encarga de desarrollar el código y supervisa a los demás programadores.
- Programador Frontend: desarrolla el software relativo a la interacción con el usuario.
- Programador Backend: desarrolla el backend del software.
- Tester: Encargado de realizar las pruebas durante el desarrollo.

Con estos recursos definidos se pueden crear las tareas y asignarlas a ellos.

Nombre de la tarea	Duración	Nombres de los recursos
Proyecto	**49 días**	
Investigación Previa	**17 días**	
Búsqueda proyectos similares	5 días	Jefe de Proyecto
Búsqueda de APIS	3 días	Jefe de Proyecto
Herramientas	**9 días**	
Python	3 días	Analista
Twitter API	1 día	Analista
Tweepy	2 días	Analista
NLTK	3 días	Analista
Análisis	**20 días**	
Requisitos	10 días	Analista
Documentación	10 días	Analista
Módulos	**9 días**	
Cuenta de Twitter	**3 días**	
Diseño	1 día	Arquitecto de Software
Implementación	3 días	Programador Frontend; Programador Senior A

Pruebas	1 día	Tester 1
Procesador de lenguaje	**3 días**	
Diseño	1 días	Arquitecto de Software
Implementación	3 días	Programador Backend; Programador Senior A
Pruebas	1 días	Tester 1
Interfaz de APIs	**3 días**	
Diseño	1 días	Arquitecto de Software
Implementación	3 días	Programador Backend; Programador Senior A
Pruebas	1 días	Tester 1
Pruebas Finales	**3 días**	
Pruebas de implementación	3 días	Tester 1
Evaluación	3 días	Analista; Arquitecto de Software; Jefe de Proyecto; Programador Senior A

Tabla 2. Planificación inicial.

De la misma forma, esto permite obtener el resumen con la duración de cada parte del desarrollo.

Nombre	Duración
Proyecto	49 días
Investigación Previa	17 días
Búsqueda proyectos similares	5 días
Búsqueda de APIS	3 días
Herramientas	9 días
Análisis	20 días
Requisitos	10 días
Documentación	10 días
Módulos	9 días
Cuenta de Twitter	3 días
Procesador de lenguaje	3 días
Interfaz de APIs	3 días
Pruebas Finales	3 días
Pruebas de implementación	3 días
Evaluación	3 días

Tabla 3. Duración por tareas.

Como se puede observar, el proyecto tendrá una duración total de 49 días laborales.

4.2 Presupuesto Inicial

Ahora, con los datos anteriores, se asignará un precio/hora a cada rol y calcularemos así el costo de la mano de obra. Primero se muestra el precio por hora de cada uno de los roles.

Nombre	Horas asignadas	Tasa estándar
Analista	256 horas	10,00 €/hora
Arquitecto de Software	48 horas	11,00 €/hora
Jefe de Proyecto	88 horas	12,00 €/hora
Programador Backend	24 horas	8,00 €/hora
Programador Frontend	48 horas	8,00 €/hora
Programador Senior A	96 horas	10,00 €/hora
Tester 1	48 horas	8,00 €/hora

Tabla 4. Coste de la mano de obra.

A continuación, veremos el coste de cada tarea planificada en la siguiente table:

Tarea	Coste
Investigación Previa	**1.488,00 €**
Búsqueda proyectos similares	480,00 €
Búsqueda de APIS	288,00 €
Herramientas	**720,00 €**
Python	240,00 €
Twitter API	80,00 €
Tweepy	160,00 €
NLTK	240,00 €
Análisis	**1.600,00 €**
Requisitos	800,00 €
Documentación	800,00 €
Módulos	**1.752,00 €**
Cuenta de Twitter	**584,00 €**
Diseño	88,00 €
Implementación	432,00 €
Pruebas	64,00 €
Procesador de lenguaje	**584,00 €**
Diseño	88,00 €
Implementación	432,00 €
Pruebas	64,00 €
Interfaz de APIs	**584,00 €**
Diseño	88,00 €
Implementación	432,00 €
Pruebas	64,00 €
Pruebas Finales	**1.224,00 €**
Pruebas de implementación	192,00 €
Evaluación	1.032,00 €
Total	**6.064,00 €**

Tabla 5. Presupuesto inicial de tareas.

A esto hay que sumarle el coste de los materiales para el desarrollo, en este caso, muchas de las tecnologías a utilizar son de licencia gratuita, y en el proyecto no se incluye el despliegue en servidor, por lo que el coste de materiales será:

Material	Precio
Ordenador	500 €
Microsoft Office	60 €
Total	**560 €**

Tabla 6. Presupuesto inicial de materiales.

Se presupone además que la empresa espera un beneficio del 20% del coste, y debemos sumarle el IVA, se muestra un desglose con el coste total.

Subtotal	**6.624,00 €**
Total, con IVA	**8.015,00 €**

Tabla 7.Presupuesto total inicial.

4.2.1 Desarrollo de Presupuesto Simplificado (Cliente)

A continuación, se muestra el presupuesto simplificado por módulo, con el beneficio prorrateado en ellos y el total sumándole el IVA.

Tareas	Coste
Investigación Previa	1.488,00 €
Análisis	1.600,00 €
Módulos	1.752,00 €
Cuenta de Twitter	584,00 €
Procesador de lenguaje	584,00 €
Interfaz de APIs	584,00 €
Pruebas Finales	1.224,00 €
Materiales	700,00 €
Subtotal	6.624,00 €
Total, Con IVA (21%)	**8.015,04 €**

Tabla 8. Presupuesto por módulos.

Capítulo 5. Evaluación y comparativa de resultados

En esta sección se discutirá la evaluación y comparativa de los distintos algoritmos de *machine learning* probados.

5.1 Elección del *Dataset*

En primer lugar, es necesario encontrar un *dataset* sobre el que entrenar el modelo, el cual debe contener una gran cantidad de mensajes clasificados por la categoría a la que pertenecen.

Para ello se realiza una búsqueda de la colección de datos en páginas como WorldBank [28], DataWorld [29] o Kaggle [30], y es en este último donde se encuentra un *dataset* que se ajusta a nuestras necesidades. Una colección que reúne preguntas de la página de Yahoo Respuestas en 4 columnas, una para la categoría, otra para la pregunta, otra para el texto que explica la pregunta y la última para la respuesta.

Esta colección es escogida ya que es la que más similitud muestra con las necesidades del proyecto, preguntas breves y concretas. Además, dado que está contenida por las 10 categorías más utilizadas de la página, que a su vez pueden utilizarse como base para la categorización que se pretende conseguir.

En términos de cuantía de datos, el *dataset* se divide en dos, la parte de entrenamiento del modelo, con 140.000 preguntas de cada categoría, resultando en un total de 1,4 millones, y la parte de *testing*, con 60.000 preguntas por cada categoría.

5.2 Evaluación del preprocesamiento

La estandarización de los datos es un proceso clave para el funcionamiento correcto de cualquier estimador basado en *machine learning*[31]. Cuanta más variabilidad en la forma en la que se presentan los datos, más probable es que el algoritmo no sepa comportarse de la forma esperada [32]. Esta estandarización se consigue mediante el preprocesamiento.

La eficiencia de este proceso depende en gran medida del algoritmo empleado para ello, por lo que, para esta parte, se llevará a cabo una investigación con el fin de decidir los pasos a seguir en el preprocesamiento [33].

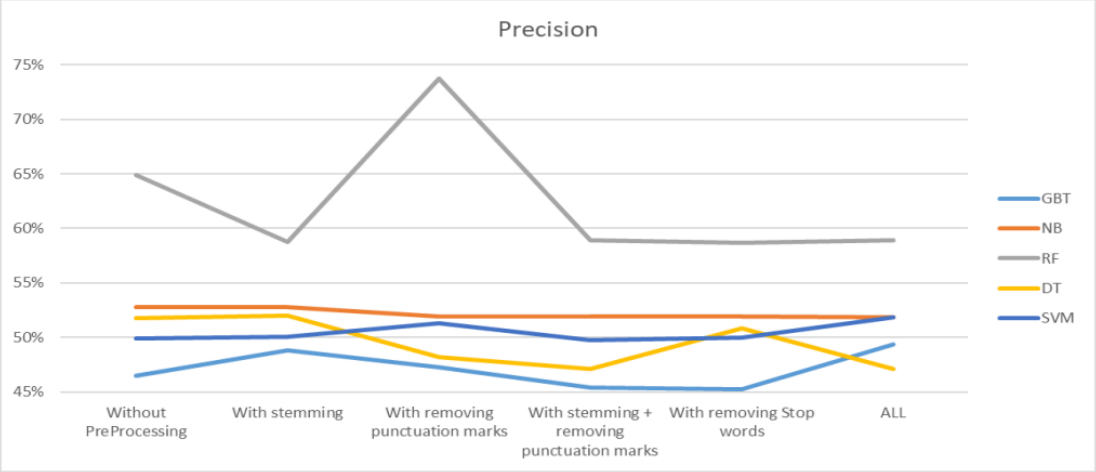

Imagen 10. Precisión según los pasos de preprocesamiento y algoritmos. Fuente: The Impact of applying Different Preprocessing Steps [26].

5.2.1 Conversión a minúsculas y eliminación de símbolos.

Dentro de la clasificación de texto es importante eliminar todas las partes que no resultan útiles para el clasificador, partes como los signos de puntuación o los caracteres especiales pueden no aportar ninguna información al modelo [34].

En este caso, tanto los signos de puntuación como las mayúsculas no aportan nada relevante de cara a saber la categoría del texto en cuestión, sin embargo, los signos de puntuación, si podrían ser útiles en otros casos, por ejemplo, si queremos saber si un texto está bien redactado o no [35].

5.2.2 Eliminación de palabras vacías

Las palabras vacías, o *stopwords*, son las más habituales en cualquier idioma, lo que quiere decir que serán las que más aparecerán tanto en nuestro *dataset* como en las preguntas de los usuarios. Son, básicamente, conjunciones, artículos, preposiciones y adverbios, palabras como "el", "la", "cuando", "desde", etc. Estos términos se eliminan con la idea de dar mayor importancia al resto de términos del texto.

En este caso se han utilizado las *Stopwords* ofrecidas por Natural Language ToolKit (NLTK) [36]. Es una lista que se puede obtener de forma muy sencilla en varios idiomas, en el caso del inglés contiene 179 palabras.

```
StopWords List
{'haven', 'but', "she's", "weren't", 'am', 'if', 'below', 'wouldn', 'my',
```

Imagen 11. Algunas de las stopwords de NLTK.

Además, otra ventaja de la eliminación de estas palabras es que se libera mucha carga de trabajo del modelo, ya que reduce en gran medida su tamaño, provocando una mejor en el rendimiento del sistema [37].

5.2.3 Lematización y *Stemming*

La lematización y el *stemming* son dos técnicas que tienen enfoques similares. El *stemming* recorta una palabra basándose en los prefijos y sufijos comunes, mientras que la lematización se basa en la morfología de la palabra, en cualquier caso, ambas dejan un lema como resultado final [38].

Stemming		
Palabra	**Sufijo**	**Lema**
studies	-es	studi
studying	-ing	study

Lematización		
Palabra	**Información Morfológica**	**Lema**
studies	Presente de study	study
studying	Infinitivo de study	study

Tabla 9. Diferencias entre stemming y lematización.

Al igual que ocurre con la eliminación de las palabras vacías, el hecho de reducir el tamaño de las palabras produce una mejora en el rendimiento. En este caso para tratar de conservar el mayor sentido morfológico, se utilizará la lematización.

5.3 Comparativa de modelos *machine learning*

Una vez seleccionados los pasos previos que se realizarán sobre nuestros datos, el siguiente paso es realizar los diferentes modelos según los algoritmos explicados previamente, y elegir cuales son los más adecuados al sistema.

5.3.1 Naive Bayes

Los primeros algoritmos que se pondrán a prueba son los Naive Bayes. Scikit-Learn ofrece varias formas de crear un modelo siguiendo este algoritmo [39].

A su vez, proporciona ciertos parámetros como aprender las prioridades de cada clase previamente a la generación del modelo, en nuestro caso, no existen prioridades en ninguna categoría [40].

La otra configuración posible es el suavizado adaptativo, que sirve para solucionar el problema que surge cuando aparece un término nuevo que no se encuentra en el *dataset* [41]. El valor por defecto es 1, siendo 0 el valor correspondiente a que no se realice ningún suavizado, así que se probará a disminuir y aumentar su valor para ver si afecta a la eficiencia.

En primer lugar, se muestran los datos obtenidos al aumentar el valor de Alpha, que es el valor asociado a la configuración del suavizado adaptativo que acabamos de explicar.

MultinomialNB				
Alpha	Precisión	*Recall*	*F1-score*	*Accuracy*
1	0,67	0,67	0,67	0,67
10	0,66	0,66	0,66	0,66
100	0,62	0,62	0,62	0,62
1000	0,54	0,54	0,54	0,54
10000	0,56	0,54	0,53	0,54

Tabla 10. Resultados MultinomialNB Alpha ascendente.

Se puede concluir que, en nuestro caso, el aumento del valor de Alpha implica una menor precisión en la clasificación.

En segundo lugar, se estudian los resultados obtenidos tras disminuir su valor.

MultinomialNB				
Alpha	**Precisión**	**Recall**	**F1-score**	**Accuracy**
1	0,67	0,67	0,67	0,67
0,1	0,66	0,66	0,66	0,66
0,001	0,66	0,66	0,66	0,66
0,00000001	0,66	0,66	0,65	0,65

Tabla 11. Resultados MultinomialNB Alpha descendente.

Aunque en menor medida, también se observa que el acierto del clasificador disminuye cuando el suavizado difiere de 1.

Por último, al eliminar por completo el suavizado, ajustando el valor a cero, se obtiene el siguiente resultado:

MultinomialNB				
Alpha	**Precisión**	***Recall***	***F1-score***	***Accuracy***
0	0,65	0,65	0,65	0,65

Tabla 12. Resultados MultinomialNB sin suavizado.

Si bien el resultado no difiere en gran medida de nuestra mejor opción, sigue siendo menos certero que originalmente, por lo que queda descartado.

Una vez concluida la configuración óptima para nuestro algoritmo, el siguiente paso es evaluar los distintos modelos que nos ofrece Scikit-Learn para Naive Bayes.

Modelo	Precisión	*Recall*	*F1-score*	*Accuracy*
MultinomialNB	0,67	0,67	0,67	0,67
ComplementNB	0,67	0,67	0,66	0,67
BernoulliNB	0,67	0,67	0,67	0,67

Tabla 13. Otros resultados Naive Bayes.

Se puede concluir que los resultados son prácticamente idénticos para cualquier modelo.

Además, el tiempo de generación de estos modelos es realmente rápido, con menos de un segundo para el entrenamiento de cada modelo.

5.3.2 Random Forest

Scikit-Learn [42] ofrece las siguientes posibilidades de configuración para el clasificador Random Forest [43]:

N_estimators: hace referencia al número de árboles de decisiones que tendrá el modelo, el valor por defecto es 100.

Criterion: especifica el criterio a seguir para escoger porque rama del árbol avanzar, puede ser Gini [44] Entropy [45] o Log *Loss* [46].

- Gini mide la frecuencia con la que cualquier elemento del *dataset* será categorizado de forma incorrecta si se le asigna una categoría aleatoriamente.
- Entropy mide el desorden de las características con la categoría objetivo.
- Log *loss* indica lo cerca que está la probabilidad de predicción del valor real correspondiente.

Max_depth: define la profundidad máxima del árbol, si no se indica ninguna, el árbol se expande hasta que las hojas son puras, es decir, cuando todos los datos de esa hoja

pertenecen a una misma clase [43]. Dada la naturaleza de los datos, expandir un árbol hasta este punto podría llevar una gran cantidad de tiempo si es que fuera posible, por lo que se empieza por una profundidad reducida que aumentará para ver cuánto mejora el rendimiento.

Combinando las diferentes opciones se obtienen los siguientes resultados:

Random Forest							
Profundidad	**N.º Arboles**	**Criterio**	**Precisión**	***Recall***	***F1-score***	***Accuracy***	**Tiempo**
5	100	Gini	0.58	0.54	0.54	0.54	2m 50s
5	100	Log *loss*	0.57	0.54	0.54	0.54	2m 55s
5	100	Entropy	0.57	0.54	0.54	0.54	2m 55s
5	200	Gini	0.59	0.56	0.56	0.56	5m 43s
5	200	Log *loss*	0.58	0.56	0.56	0.56	5m 44s
5	200	Entropy	0.58	0.56	0.56	0.56	5m 43s
5	300	Gini	0.59	0.57	0.57	0.57	8m 34s
5	300	Log *loss*	0.58	0.57	0.57	0.57	8m 31s
5	300	Entropy	0.58	0.57	0.57	0.57	8m 29s
10	100	Gini	0.58	0.54	0.54	0.54	5m 22s
10	100	Log *loss*	0.57	0.54	0.54	0.54	5m 20s
10	100	Entropy	0.57	0.54	0.54	0.54	5m 34s
10	200	Gini	0.59	0.56	0.56	0.56	10m 59s
10	200	Log *loss*	0.58	0.56	0.56	0.56	11m 01s
10	200	Entropy	0.58	0.56	0.56	0.56	11m 00s
10	300	Gini	0.59	0.57	0.57	0.57	16m 04s
10	300	Log *loss*	0.58	0.57	0.57	0.57	16m 12s
10	300	Entropy	0.58	0.57	0.57	0.57	15m 48s
20	100	Gini	0.58	0.54	0.54	0.54	11m 06s
20	100	Log *loss*	0.57	0.54	0.54	0.54	11m 40s
20	100	Entropy	0.57	0.54	0.54	0.54	11m 48s
20	200	Gini	0.59	0.56	0.56	0.56	23m 11s
20	200	Log *loss*	0.58	0.56	0.56	0.56	24m 59s
20	200	Entropy	0.58	0.56	0.56	0.56	25m 29s
20	300	Gini	0.59	0.57	0.57	0.57	34m 41s
20	300	Log *loss*	0.58	0.57	0.57	0.57	34m 12s
20	300	Entropy	0.58	0.57	0.57	0.57	35m 45s

Tabla 14. Resultados Random Forest.

Evaluando los resultados se aprecia que aumentar el número de árboles mejora todos los aspectos ligeramente. Con respecto a los criterios, Gini parece ser la mejor opción en cualquiera de los escenarios. La profundidad no parece afectar positiva ni negativamente al resultado con estos valores. Posteriormente, se evaluará si al aumentarla más significativamente mejora el resultado.

Hay que tener en cuenta que, tanto el aumento de la profundidad, como del número de árboles aumenta significativamente el tiempo necesario para generar y entrenar el modelo.

Por último, una vez elegida la mejor configuración posible, se comprobará si existen mejoras en el resultado al aumentar la profundidad a un valor de 100.

Random Forest						
Profundidad	**Nº Arboles**	**Criterio**	**Precisión**	***Recall***	***F1-score***	***Accuracy***
100	300	Gini	0.62	0.60	0.60	0.60

Tabla 15. Resultados Random Forest Profundidad 100.

Efectivamente, se afirma que la profundidad del árbol nos permite subir la certeza del modelo si aumenta de forma considerable, si bien hace que el tiempo para generarlo se incremente proporcionalmente, alcanzando hasta las 3 horas.

5.3.3 Decision Tree

El algoritmo de Decision Tree, salvo que se especifique lo contrario, trabaja para realizar un árbol con las hojas puras, es decir, hasta que todos los datos en ellas pertenezcan a una sola clase [47], en nuestro caso, eso requerirá de un elevado tiempo de computación, y reducir la profundidad como se realizó anteriormente con Random Forest aseguraría unos peores resultados, ya que RF utiliza varios árboles de decisiones y aquí tan solo generamos uno. Con esto, se prueba a cambiar únicamente el criterio.

Decision Tree				
Criterio	**Precisión**	***Recall***	***F1-score***	***Accuracy***
Gini	0,54	0,53	0,53	0,53
Entropy	0,53	0,52	0,52	0,52
Log *Loss*	0,53	0,52	0,52	0,52

Tabla 16.Resultados Decision Tree.

Se aprecia que los resultados obtenidos no mejoran a los del algoritmo Random Forest, si bien al igual que en este, el criterio Gini parece el más adecuado para nuestro trabajo. En términos de rendimiento, para la generación de cada uno de estos modelos se tardó, de media, 1 hora y 50 minutos, con los tres criterios tardando tiempos parecidos.

5.3.4 LinearSVC

Las configuraciones posibles para este algoritmo son *penalty*, *loss* y *dual*, las cuales tienen que ver con la forma en la que se resuelve la ecuación de optimización [48]. Hay que apuntar que no todas las combinaciones se permiten, porque Scikit-Learn, por lo menos por el momento, no las ha implementado [49].

Linear SVC							
Penalty	***Loss***	***Dual***	**Precisión**	***Recall***	***F1-score***	***Accuracy***	**Tiempo**
l1	Squared_hinge	TRUE	No es posible				
l1	Squared_hinge	TRUE	No es posible				
l1	Squared_hinge	FALSE	0,68	0,68	0,67	0,68	2min
l1	Hinge	TRUE	No es posible				
l1	Hinge	FALSE	No es posible				
l2	Squared_hinge	TRUE	0,68	0,68	0,67	0,68	30 min
l2	Squared_hinge	FALSE	0,68	0,68	0,67	0,68	30 min
l2	Hinge	TRUE	0,67	0,67	0,66	0,67	30 min
l2	Hinge	FALSE	No es posible				

Tabla 17.Resultados LinearSVC.

En los resultados no se aprecian prácticamente las diferencias entre las configuraciones. Sin embargo, sí es apreciable la diferencia de tiempo a la hora de generar los modelos, ya que la estrategia L1 apenas tarda dos minutos mientras que las de L2 se extiende hasta media hora.

Otra configuración disponible es el número de iteraciones que realiza el modelo hasta considerarse entrenado, esto puede afectar significativamente al tiempo en el que se genera el modelo, por lo que una vez se haya escogido uno, se probará a aumentarlo de 1000, el valor por defecto, a 4000.

Linear SVC 4000 Iteraciones						
Penalty	***Loss***	***Dual***	**Precisión**	***Recall***	***F1-score***	***Accuracy***
l2	Squared_hinge	FALSE	0,68	0,68	0,67	0,68

Tabla 18. Resultados Linear SVC aumento iteraciones.

Se aprecia que el resultado no se ve alterado por aumentar las iteraciones.

5.3.5 KNeighbors

El algoritmo KNeighbors permite configurar el número de vecinos que se utilizarán, la forma de calcular el peso que tendrá cada uno de ellos en la predicción y el algoritmo que definirá cuales son los más cercanos al punto a predecir [50]. En primer lugar, modificaremos las dos configuraciones relativas al peso.

KNeighbors					
Weight	**Precisión**	***Recall***	***F1-score***	***Accuracy***	**Tiempo**
Uniform	0,48	0,46	0,47	0,46	3 horas
Distance	0,51	0,48	0,48	0,48	3 horas

Tabla 19. Resultados KNeighbors.

Como se puede observar, el criterio en el que se toma como influyente la distancia de los vecinos al punto a predecir, en lugar de considerarlos todos de forma uniforme, funciona mejor para nuestros datos.

La otra configuración que se modificará será el algoritmo para buscar los puntos.

KNeighbors				
Algorithm	**Precisión**	***Recall***	***F1-score***	***Accuracy***
Auto	0,51	0,48	0,48	0,48
Brute-force	0,51	0,48	0,48	0,48

Tabla 20. Resultados KNeighbors.

No se aprecian diferencias ni en la precisión ni en el tiempo que se tarda en generarlo.

5.3.6 Comparativa de modelos

Una vez hechos todos los modelos, en este subapartado se comparan para decidir cuál es el óptimo para nuestro proyecto.

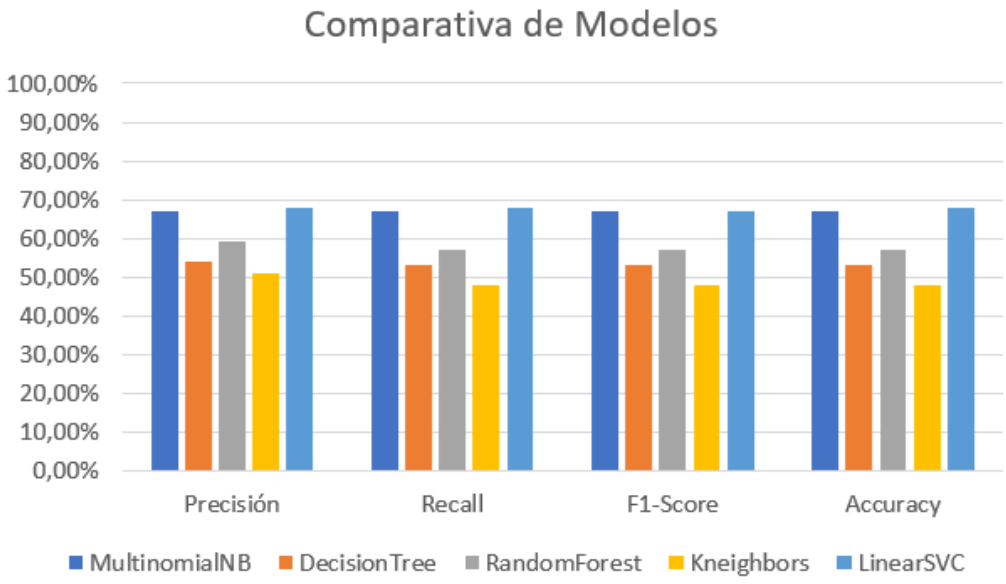

Imagen 12. Comparativa de modelos.

En base a estos datos se decide seleccionar el modelo creado mediante Naive Bayes y el LinearSVC para el siguiente paso.

5.4 Stacking

Stacking es el proceso de apilar distintos modelos previamente entrenados con el fin de elegir una sola predicción [51]. Para combinar los distintos modelos se usa un clasificador final, por

defecto Scikit-Learn utiliza un LogisticRegression [52], un proceso automatizado por SKLearn, se genera y entrena como cualquier otro clasificador.

Se evalúan, por tanto, los resultados obtenidos al combinar MultinomialNB y LinearSVC, los dos modelos que mejor resultado han dado.

Stacking			
Precisión	***Recall***	***F1-score***	***Accuracy***
0,68	0,68	0,68	0,68

Tabla 21. Resultados Stacking SKLearn.

Además, se prueba a realizar un *stacking* manual. Para ello, cuando llegue una predicción que hacer, se realizará con los dos modelos y será elegida la que mayor probabilidad dé a la categoría escogida por el modelo, mediante el método *predict_proba* [53]. Por la forma en la que SKLearn ha creado el método *classification_report*, es posible generar el informe con los resultados de nuestro experimento, si se realiza esto para todos los casos de test y se compara con los resultados esperados.

***Stacking* Manual**			
Precisión	***Recall***	***F1-score***	***Accuracy***
0,68	0,68	0,67	0,68

Tabla 22. Resultados Stacking Manual.

Como se puede observar, los resultados apenas difieren de los originales.

Comparando ahora todos los resultados obtenidos se obtiene la siguiente gráfica:

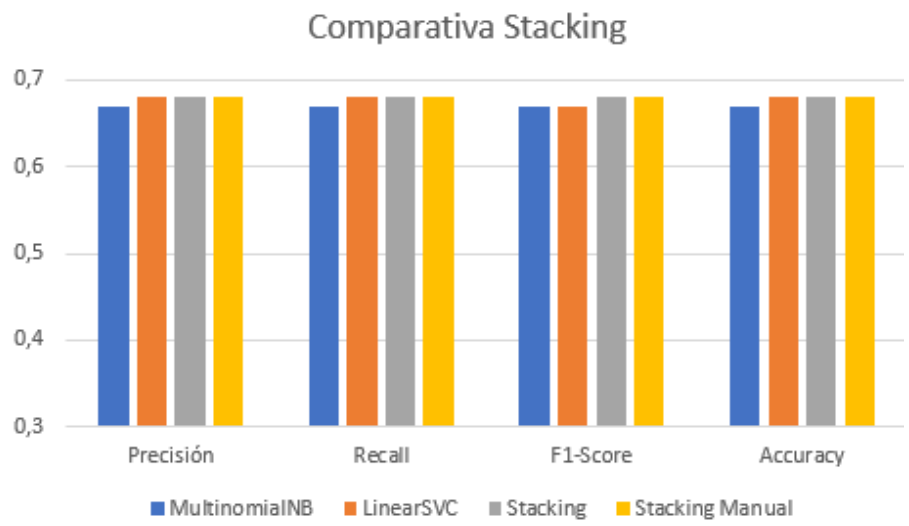

Imagen 13. Comparativa Stacking.

Se puede apreciar que el *Stacking* proporciona una muy leve mejoría con respecto a nuestros mejores modelos.

Capítulo 6. Análisis

Este apartado contendrá toda la especificación de requisitos y toda la documentación del análisis de la aplicación, a partir de la cual se elaborará posteriormente el diseño.

6.1 Definición del Sistema

En este capítulo se definirá el sistema, determinando su alcance.

6.1.1 Determinación del Alcance del Sistema

El sistema, como se definió en el punto *Objetivos del Proyecto* debe ser capaz de responder a las preguntas que cualquier usuario haga vía Twitter. Es necesario, por tanto, ser capaces de analizar el texto de las preguntas que nos realizan, y posteriormente de buscar una respuesta acorde.

Para el análisis del texto el foco será categorizarlo para así poder hacer una búsqueda específica posterior. La forma que se ha elegido para dicha categorización es la clasificación de texto basada en *machine learning* [54] Así, se selecciona el inglés como idioma en el que deben hacerse los *tweet*s, debido a que es el idioma dominante en lo que a recursos en este campo se refiere.

Dado que se pretende clasificar, primero se deben establecer las categorías a obtener y posteriormente, que lo que se hará con cada una de ellas.

Las categorías obtenidas en nuestro *dataset* son las que utilizaba la página Yahoo Respuestas para clasificar sus preguntas: “Sociedad y Cultura”, “Ciencia, Salud”, “Educación”, “Informática e Internet”, “Deportes”, “Negocios y Finanzas”, “Música y Entretenimiento”, “Familia y Relaciones” y “Política y Gobierno”.

Una vez establecidas las categorías, para buscar las respuestas se decidió utilizar:

- La API de Wikipedia para las preguntas de Ciencia y Matemáticas, Historia, Educación, y Política y Gobierno.
- Un motor de búsqueda de Google que busque exclusivamente en la página de NHS (*National Health Service*), el servicio de salud pública de Reino Unido.
- Una búsqueda con el motor estándar de Google para el resto

.

6.2 Requisitos del Sistema

En esta sección se tratarán los requisitos, actores, subsistemas y casos de uso del sistema.

6.2.1 Obtención de los Requisitos del Sistema

A continuación, se listan los requisitos que tendrá nuestro sistema:

Código	Nombre Requisito	Descripción del Requisito
RTW.1	Detectar un *tweet*	El sistema debe ser capaz de detectar que un usuario de Twitter le ha mencionado en un *tweet*.
RTW.2	Recoger texto de un *tweet*	El sistema debe poder extraer el texto de un *tweet* en el que se le menciona.
RTW.3	Detectar que un *tweet* ha sido respondido	Si ya se ha respondido previamente a un *tweet*, el sistema lo debe saber y no deber volver a hacerlo.
RTW.4	Enviar un *tweet* de respuesta	Debemos ser capaces de publicar *tweet*s como respuesta a las menciones de los usuarios.
RPR.1	Texto en minúsculas	El texto que recibimos debemos pasarlo a minúsculas.
RPR.2	Eliminar las palabras vacías	Al texto que recibimos se le eliminarán las palabras vacías.
RPR.3	Lematizar el texto	Debemos lematizar el texto recibido.
RCLF.1	Clasificar texto	El sistema debe clasificar el texto.
RCLF.1.1	Clasificar pregunta como Salud	El sistema debe ser capaz de clasificar una pregunta como Salud.
RCLF.1.2	Clasificar pregunta como Sociedad y Cultura	El sistema debe ser capaz de clasificar una pregunta como Sociedad y Cultura.
RCLF.1.3	Clasificar pregunta como Educación	El sistema debe ser capaz de clasificar una pregunta como Educación.
RCLF.1.4	Clasificar pregunta como Deportes	El sistema debe ser capaz de clasificar una pregunta como Deportes.
RCLF.1.5	Clasificar pregunta como Informática	El sistema debe ser capaz de clasificar una pregunta como Informática.
RCLF.1.6	Clasificar pregunta como Negocios y finanzas	El sistema debe ser capaz de clasificar una pregunta como Negocios y Finanzas.
RCLF.1.7	Clasificar pregunta como Entretenimiento y música	El sistema debe ser capaz de clasificar una pregunta como Entretenimiento y música.

RCLF.1.8	Clasificar pregunta como Familia y relaciones	El sistema debe ser capaz de clasificar una pregunta como Familia y relaciones.
RCLF.1.9	Clasificar pregunta como Política	El sistema debe ser capaz de clasificar una pregunta como Política.
RCLF.1.10	Clasificar pregunta como Ciencia	El sistema debe ser capaz de clasificar una pregunta como Ciencia.
RB.1	Utilizar la API correspondiente a la pregunta	En función del tipo de pregunta que se haya realizado el sistema debe ser capaz de asignarle una API acorde.
RB.1.1	Asignar API de Wikipedia	El sistema debe asignar esta API a las preguntas de Ciencia, Educación, Política y Finanzas.
RB.1.2	Asignar API de NHS	El sistema asignará esta API a las preguntas de Salud.
RB.1.3	Asignar API de Google	El sistema asignará esta API a las categorías restantes.
RB.2	Búsqueda en Wikipedia	El sistema debe ser capaz de buscar la pregunta lanzada en la API de Wikipedia.
RB.3	Búsqueda en NHS	El sistema debe ser capaz de buscar la pregunta lanzada en la página de NHS.
RB.4	Búsqueda en Google	El sistema debe ser capaz de buscar la pregunta lanzada en el motor de búsqueda estándar de Google.
RB.5	Enlace de respuesta	El sistema debe dar también un enlace donde obtener más información sobre la respuesta.

Tabla 23. Requisitos del sistema.

6.2.2 Identificación de Actores del Sistema

En nuestro sistema se encuentra solo un tipo de actores primarios, siendo estos los usuarios de Twitter (registrados) que realizan una pregunta a nuestra cuenta. Los usuarios anónimos (no registrados) podrán ver las respuestas del sistema a otros usuarios, pero no realizar propias.

En cuanto a actores secundarios aparecen cuatro:

- La API de Twitter, que se utiliza tanto para extraer el texto de las preguntas como para mandar una respuesta.
- La base de datos utilizada para entrenar los modelos.

- La API de Wikipedia, para responder a ciertos tipos de preguntas.
- El API de Google, la cual se utilizará para acceder a los distintos motores de búsqueda creados para responder al resto de preguntas.

6.2.3 Especificación de Casos de Uso

A continuación, se definirán los casos de uso de nuestro sistema, se han encontrado los siguientes:

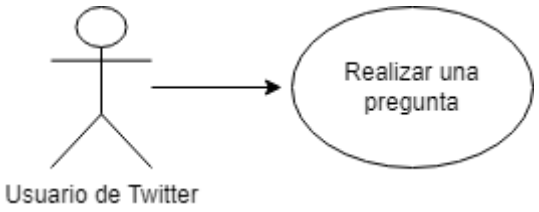

Imagen 14. Casos de uso para los usuarios de Twitter.

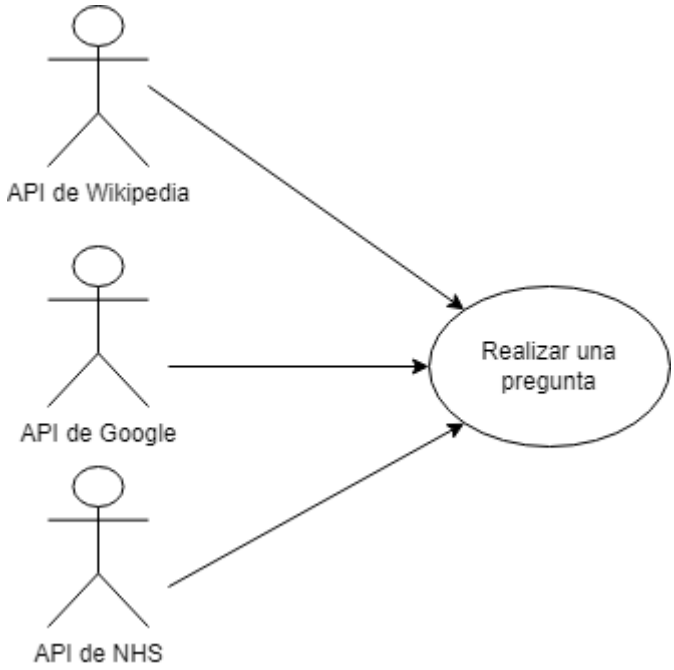

Imagen 15. Casos de uso para las APIs.

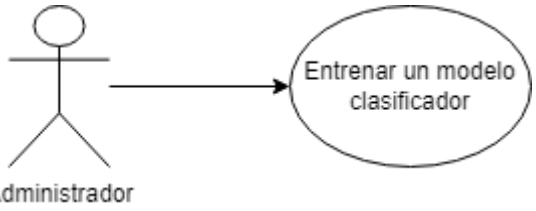

Imagen 16. Casis de uso para el administrador.

6.2.3.1 Caso de uso 1: Realizar una pregunta

Nombre del Caso de Uso
Realizar una pregunta
Descripción
El usuario de Twitter debe ser capaz de realizar una pregunta, mencionando en un *tweet* a la cuenta de nuestro sistema, el sistema debe ser capaz de recoger el texto de la pregunta, preprocesarlo y predecir la categoría a la que pertenece.

Tabla 24 . Caso de uso Realizar una pregunta.

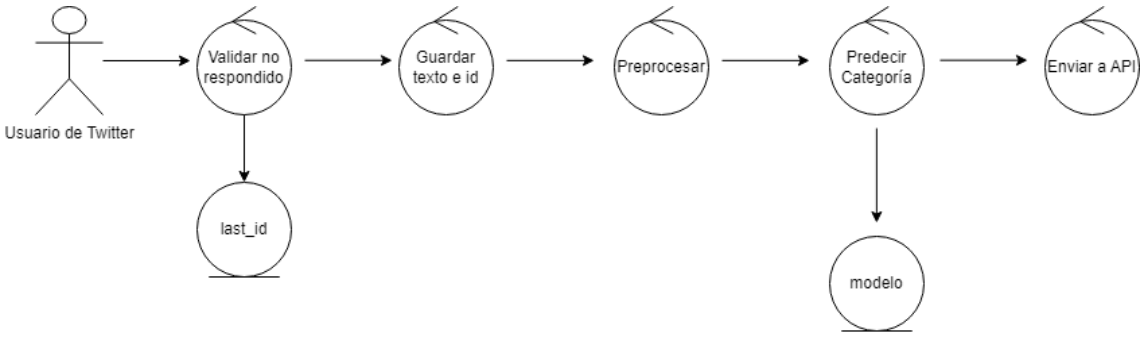

Imagen 17. Responder a una pregunta con la API de Wikipedia.

6.2.3.2 Caso de uso 2: Responder a una pregunta con la API de Wikipedia

Nombre del Caso de Uso
Responder a una pregunta con la API de Wikipedia
Descripción
El sistema debe ser capaz de utilizando la API de Wikipedia, obtener una respuesta a la cuestión, extraer el enlace a la página correspondiente y enviar un *tweet* con la información

Tabla 25. Responder a una pregunta con la API de Wikipedia

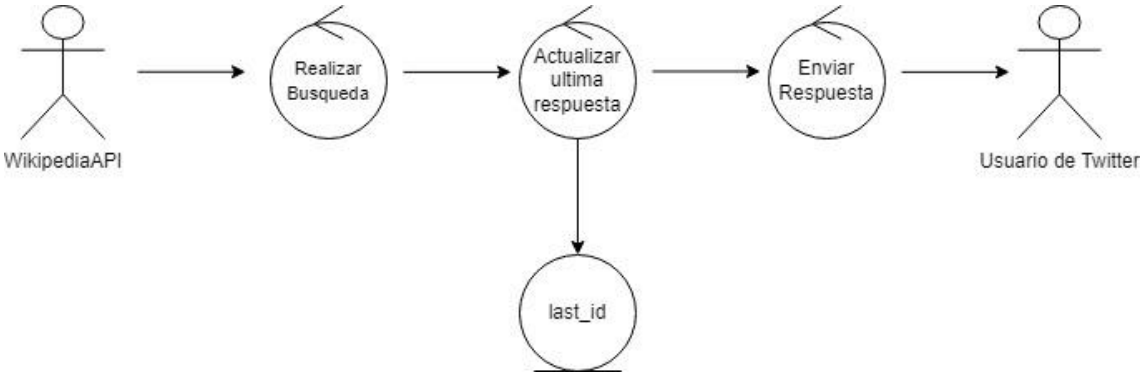

Imagen 18. Responder a una pregunta con la API de Wikipedia.

6.2.3.3 Caso de uso 3: Responder a una pregunta con NHS

Nombre del Caso de Uso
Responder a una pregunta con NHS
Descripción
El sistema debe ser capaz de utilizando el motor de búsqueda personalizado sólo con la página de NHS, obtener una respuesta a la cuestión, extraer el enlace a la página correspondiente y enviar un *tweet* con la información

Tabla 26. Responder a una pregunta con NHS

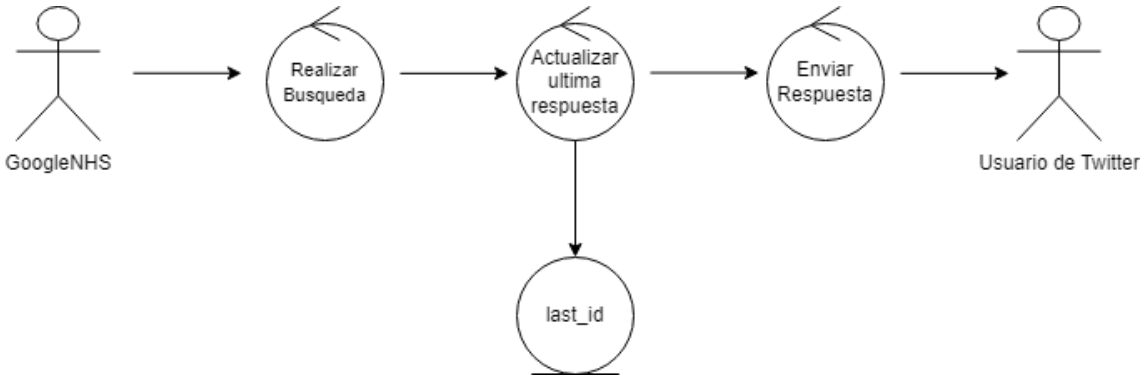

Imagen 19. Responder a una pregunta con NHS.

6.2.3.4 Caso de uso 4: Responder a una pregunta con la API de Google

Nombre del Caso de Uso
Responder a una pregunta con la API de Google
Descripción

El sistema debe ser capaz de utilizando el motor de búsqueda general de la API de Google, obtener una respuesta a la cuestión, extraer el enlace a la página correspondiente y enviar un *tweet* con la información

Tabla 27. Responder a una pregunta con la API de Google.

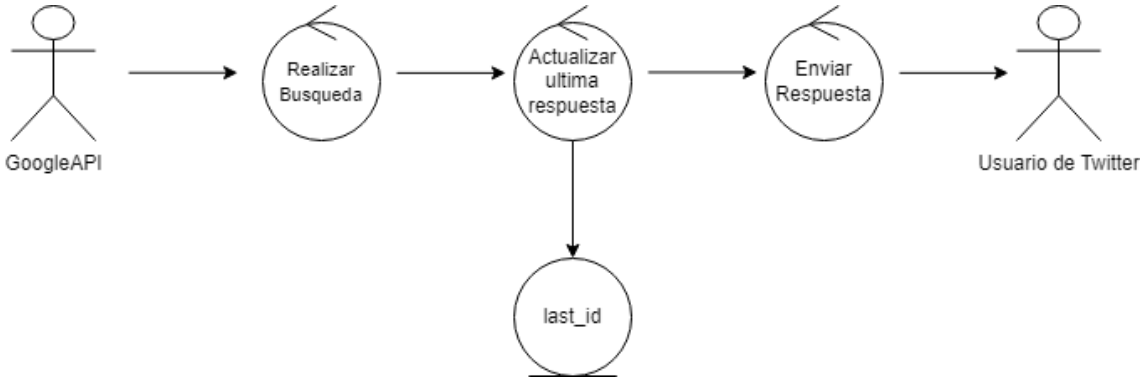

Imagen 20. Responder a un usuario usando la API de Google.

6.2.3.5 Caso de uso 5: Entrenar un modelo

Nombre del Caso de Uso
Entrenar un modelo
Descripción
El administrador deberá entrenar un modelo, buscando un *dataset* acorde y almacenarlo localmente para su uso futuro en el sistema

Tabla 28. Responder a una pregunta con la API de Google.

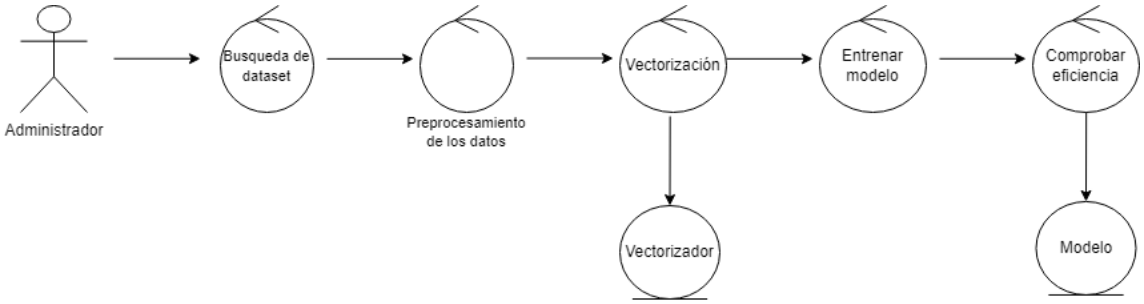

Imagen 21. Entrenar un modelo.

6.3 Identificación de los Subsistemas en la Fase de Análisis

El objetivo de esta sección es analizar el sistema para poder descomponerlo en sistemas más pequeños (subsistemas) que faciliten su posterior análisis.

6.3.1 Descripción de los Subsistemas

En nuestro proyecto se encuentran los siguientes subsistemas:

- **Interfaz de Twitter:** aquí se realiza todo lo relacionado con la red social, las preguntas de los usuarios que no han sido respondidas son recopiladas y una vez el resto del sistema ha conseguido la respuesta acorde, se responde al *tweet* original con ella.
- **Preprocesamiento:** este módulo es el encargado de tratar el texto de la pregunta para que esté estandarizada de la forma que el modelo espera.
- **Clasificador:** aquí el texto se procesa con el modelo creado para la categorización y se obtiene una categoría de pregunta.
- **Módulo de respuesta:** en función de la categoría de la pregunta, en este subsistema se utilizan diferentes motores de búsqueda o APIs para obtener el texto de respuesta, así como el enlace para obtener más información.

6.3.2 Descripción de los Interfaces entre Subsistemas

Los subsistemas anteriormente mencionados se comunicarán, todos localmente, de la siguiente forma:

El módulo de Twitter envía el texto de la pregunta al preprocesador, clasificador y finalmente al módulo de respuesta, el cual devuelve la información necesaria para responder.

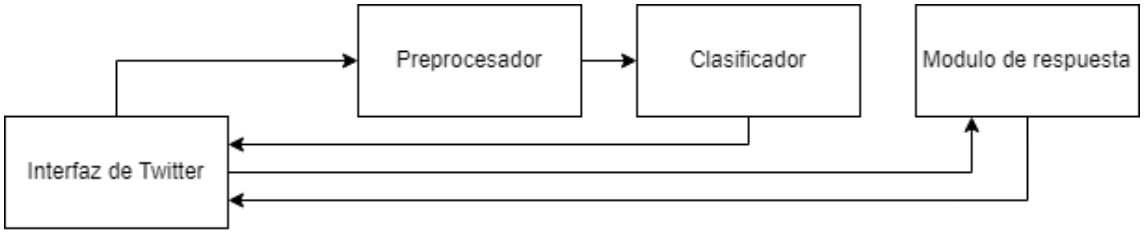

Imagen 22. Interfaces entre subsistemas.

6.4 Diagrama de Clases Preliminar del Análisis

En esta sección se describen las clases y se muestra su diagrama por subsistemas.

6.4.1 Diagrama de Clases

Aquí se exponen las clases que compondrán cada subsistema.

6.4.1.1 Subsistema de Interfaz de Twitter

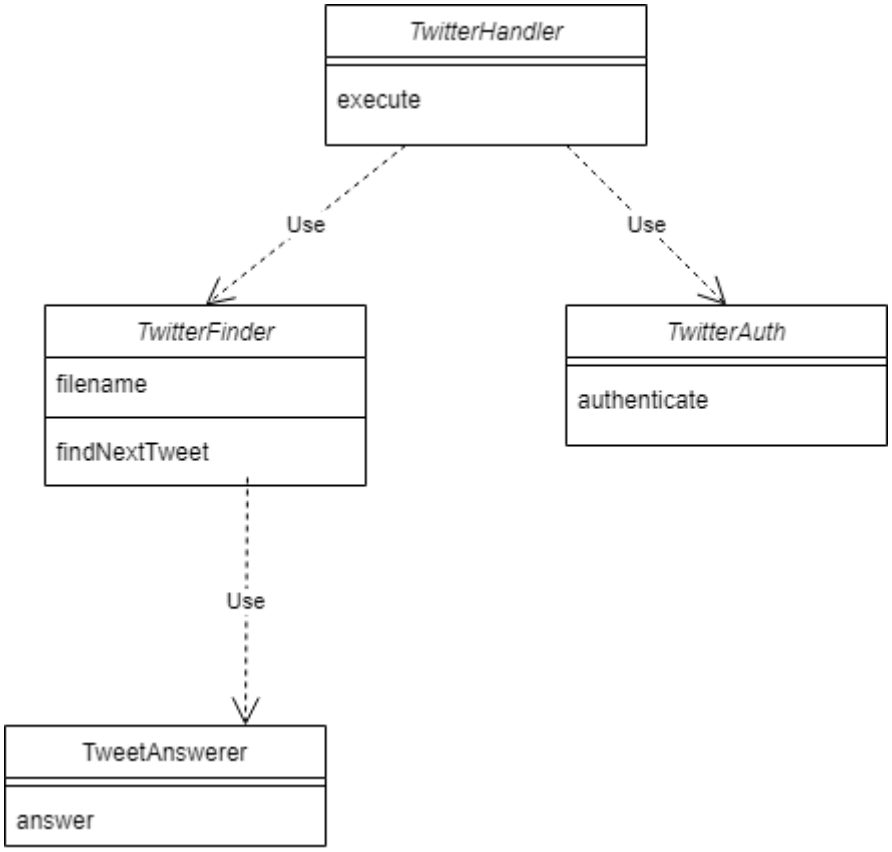

Imagen 23. Subsistema de Interfaz de Twitter.

En este subsistema se incluyen 4 clases:

- **TwitterHandler** será la encargada de la ejecución.
- ***TwitterAuth*** se encargará de la autenticación.
- **TwitterFinder** es la clase encargada de conseguir el siguiente *tweet* que debe ser respondido.
- ***TweetAnswerer*** tendrá la función de responder a los *tweet*s de los usuarios.

6.4.1.2 Subsistema de Preprocesamiento

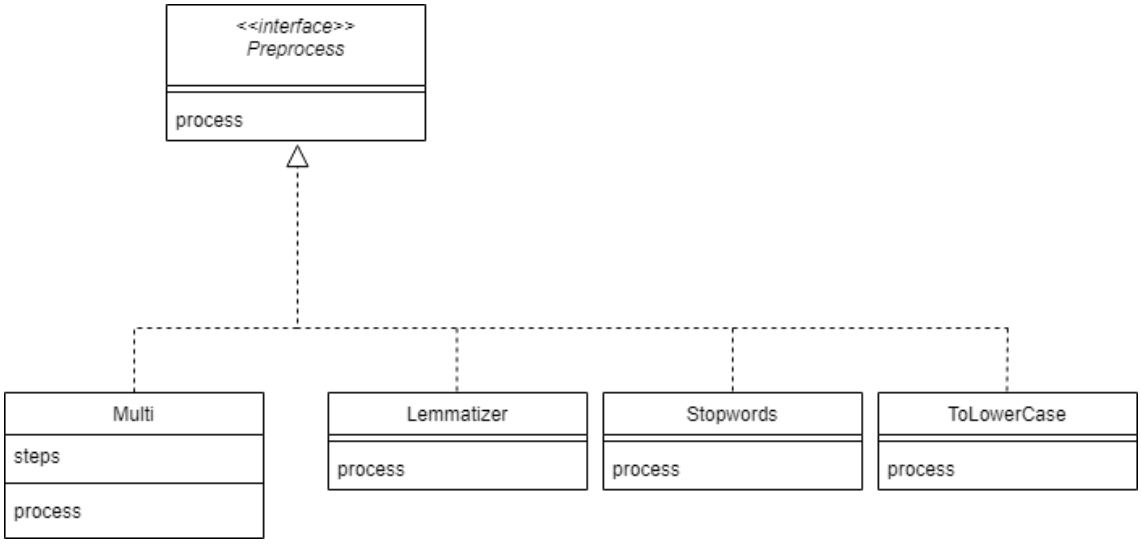

Imagen 24. Subsistema de Preprocesamiento.

En el subsistema de preprocesamiento se encuentra una clase por cada paso de este, siguiendo un patrón *Strategy* [55] para la creación de los pasos y después un *Composite* [56] como solución para cuando se quieran realizar varios pasos de preprocesamiento.

6.4.1.3 Subsistema de Clasificación

Este subsistema será una única clase, ya que la creación del modelo capaz de hacer la clasificación se hará previamente y solo debemos cargarlo y realizar la predicción.

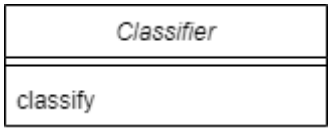

Imagen 25. Subsistema de Clasificación.

6.4.1.4 Subsistema de Respuesta

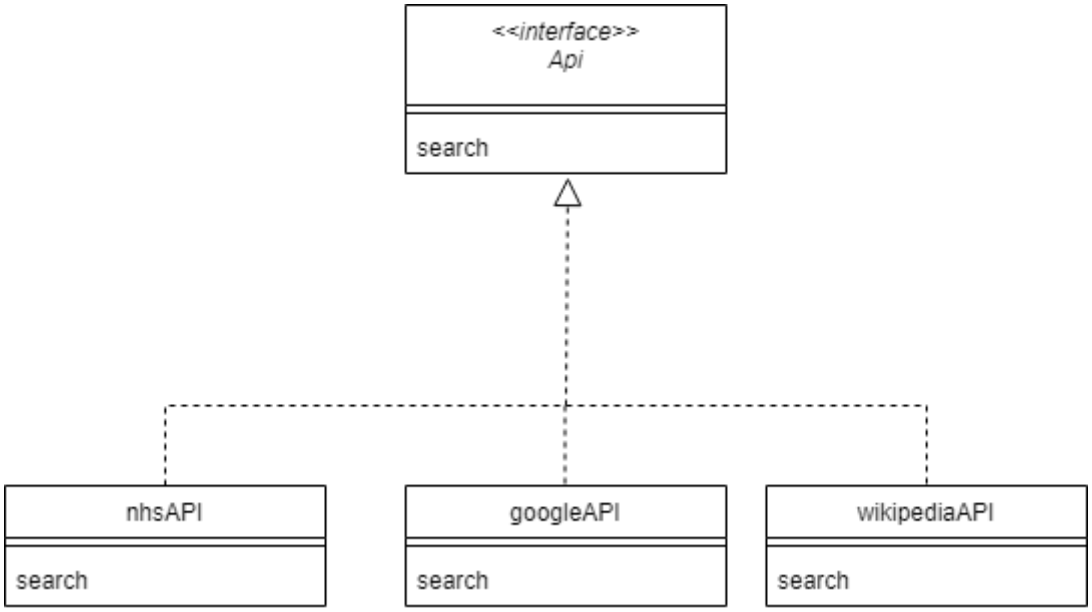

Imagen 26. Subsistema de respuesta.

En este subsistema se creará un patrón *Strategy*, y cada estrategia concreta será un *Adapter*[57] de su respectiva API externa.

6.4.2 Descripción de las Clases

En esta sección se describirán en profundidad las clases mencionadas anteriormente:

6.4.2.1 Interfaz de Twitter

Nombre de la Clase
Twitter*Answer*er
Descripción
Esta clase será la encargada de responder a los *tweet*s con las preguntas.
Responsabilidades
Recibe el texto y el id del *tweet*, lo envía al resto de subsistemas que se encargan de encontrar la respuesta. Una vez tiene la respuesta, responde al *tweet* del usuario con ella y el enlace para obtener más información.
Atributos Propuestos
.

Métodos Propuestos
answer: Recibe el id del *tweet* y el texto de la pregunta, llama a los métodos correspondientes de preprocesamiento, clasificación y búsqueda de respuesta, y responde al *tweet* del usuario con la respuesta obtenida.

Tabla 29. Clase TwitterAnswerer.

Nombre de la Clase
TwitterFinder
Descripción
Esta clase será la encargada de buscar el siguiente *tweet* a responder.
Responsabilidades
Deberá ser capaz de recordar el id del último *tweet* que se ha respondido, para así mandar todos los siguientes no respondidos a la clase Twitter*Answer*er. Además, deberá guardar localmente el id del último *tweet* una vez haya finalizado con todos, para que se tenga en la siguiente ejecución.
Atributos Propuestos
last_id: El id del último *tweet* que se ha respondido, así cuando recibamos la lista de *tweet*s sabremos cuando parar de responder.
Métodos Propuestos
findNext*Tweet*: Deberá recopilar todas las menciones de la cuenta, y empezando por la más reciente enviar el texto al método correspondiente de *TwiterAnswerer*. Además, una vez terminado almacenará en last_id el id de la mención más reciente.

Tabla 30. Clase TwitterFinder.

Nombre de la Clase
TwitterHandler
Descripción
Esta clase reunirá al resto de clases del subsistema.
Responsabilidades
Llamará a los métodos de autenticación de la clase *TwitterAuth*, y se encargará de llamar periódicamente al método que nos busca el siguiente *tweet* a responder de la clase *TwitterFinder*.

Atributos Propuestos
.
Métodos Propuestos
Execute: Será el encargado de llamar al método de autenticación de *TwitterAuth* y en bucle, llamará al método *findNextTweet* de *TwitterFinder*.

Tabla 31. Clase TwitterHandler.

6.4.2.2 Subsistema de Preprocesamiento

Nombre de la Clase
PreProcess
Descripción
Interfaz que define los métodos que deben tener los pasos de preprocesamiento.
Responsabilidades
Definirá el contrato que seguirás todos los pasos.
Atributos Propuestos
.
Métodos Propuestos
***process*:** ejecuta el preprocesamiento del paso en el que nos encontremos.

Tabla 32. Interfaz PreProcess.

Nombre de la Clase
Stopwords
Descripción
Paso que elimina las palabras vacías del texto.
Responsabilidades
Eliminará las palabras vacías de un texto dado como parámetro.
Atributos Propuestos
.

Métodos Propuestos
***process*:** ejecuta el preprocesamiento y devuelve el texto procesado.

Tabla 33. Clase Stopwords

Nombre de la Clase
ToLowerCase
Descripción
Pone el texto en minúscula.
Responsabilidades
Devolverá en minúscula un texto dado como parámetro.
Atributos Propuestos
.
Métodos Propuestos
***process*:** ejecuta el preprocesamiento y devuelve el texto procesado.

Tabla 34. Clase ToLowerCase.

Nombre de la Clase
Multi
Descripción
Esta clase nos permite seguir un patrón *composite* con los pasos del preprocesamiento, permitiendo que nuestro sistema no tenga que diferenciar entre hacer un único paso o varios.
Responsabilidades
Recibe una lista de pasos que se quieren seguir y llama al *process* de cada uno.
Atributos Propuestos
steps: lista de pasos que deben ejecutarse.
Métodos Propuestos
***process*:** llama al *process* de cada elemento de la lista

Tabla 35. Clase Multi.

6.4.2.3 Subsistema de Clasificación

Nombre de la Clase
Classifier
Descripción
Clase encargada de obtener la categoría de una pregunta.
Responsabilidades
Recibe el texto preprocesado y realiza una predicción con el modelo entrenado para ello.
Atributos Propuestos
.
Métodos Propuestos
classify: Recibe el texto como parámetro, carga el modelo que esta guardado localmente y devuelve la predicción realizada por este.

Tabla 36. Clase Classifier.

6.4.2.4 Subsistema de Respuesta

Nombre de la Clase
Api
Descripción
Interfaz que define los métodos que deben tener las clases que adaptaran las distintas APIs.
Responsabilidades
Definirá el contrato que seguirás todas las clases que adapten las APIs.
Atributos Propuestos
.
Métodos Propuestos
***search*:** ejecuta la búsqueda.

Tabla 37. Interfaz Api

Nombre de la Clase

WikipediaAPI
Descripción
Clase que servirá para adaptar nuestra interfaz y la API de Wikipedia para obtener una respuesta.
Responsabilidades
Obtiene una respuesta utilizando la API de Wikipedia.
Atributos Propuestos
.
Métodos Propuestos
***search*:** ejecuta la búsqueda en la API y devuelve el texto resultado y el enlace a la página correspondiente de Wikipedia.

Tabla 38. Clase WikipediaAPI

Nombre de la Clase
GoogleAPI
Descripción
Clase que servirá para adaptar nuestra interfaz y nuestro motor de búsqueda de Google general.
Responsabilidades
Obtiene una respuesta utilizando el motor de búsqueda general de Google.
Atributos Propuestos
.
Métodos Propuestos
***search*:** ejecuta la búsqueda en la API de Google y devuelve el texto resultado y el enlace a la página correspondiente.

Tabla 39. Clase GoogleAPI

Nombre de la Clase
NHSAPI
Descripción

Clase que servirá para adaptar nuestra interfaz y nuestro motor de búsqueda de Google específico para la página de NHS.
Responsabilidades
Obtiene una respuesta utilizando el motor de búsqueda que hemos creado para NHS.
Atributos Propuestos
.
Métodos Propuestos
***search*:** ejecuta la búsqueda en la API de Google y devuelve el texto resultado y el enlace a la página de NHS correspondiente.

Tabla 40. Clase NHSAPI.

6.5 Análisis de Casos de Uso y Escenarios

Analizaremos en detalle los casos de uso:

6.5.1 Realizar una pregunta

Realizar una pregunta	
Precondiciones	El usuario debe tener una cuenta registrada y no protegida, es decir, con los *tweet*s visibles para todos.
Poscondiciones	No hay poscondiciones.
Actores	El usuario de Twitter, la interfaz de Twitter, el modelo clasificador
Descripción	El usuario escribirá un t*weet* mencionando a la cuenta de nuestro sistema, que comprobará primero que el t*weet* no ha sido respondido, es decir, su id no coincide con el atributo last_id. Del *tweet* se extraerá el texto que será preprocesado, y enviado para su clasificación mediante el modelo previamente entrenado.
Variaciones (escenarios secundarios)	.
Excepciones	Si se diese el caso que el archivo donde se almacena el ultimo id que se ha respondido se borrase, el sistema comprobaría en su lugar que el *tweet* que está recogiendo se envió posteriormente al encendido del sistema, para asegurarnos de que no se responderá a el mismo *tweet* dos veces.
Notas	

Tabla 41. Caso de uso detallado Realizar una pregunta.

6.5.2 Responder a una pregunta con la API de Wikipedia

Responder a una pregunta con la API de Wikipedia	
Precondiciones	El sistema debe tener el texto de la pregunta y el id del *tweet* en el que se realiza, y la categoría predicha en el Caso de uso 1 debe ser

	Ciencia, Historia, Política o Educación.
Poscondiciones	El ultimo id respondido se actualiza al del *tweet* que acabamos de responder y el usuario recibe un *tweet* de respuesta.
Actores	La interfaz de Twitter, el usuario de Twitter, el modelo de clasificación y la API de Wikipedia.
Descripción	El sistema utilizará el texto de la pregunta y la API para obtener una respuesta, guardará dicha información y el enlace correspondiente de Wikipedia, después mediante Twitter*Answer*er responderá al usuario y almacenará el id del *tweet* respondido para asegurarse de no duplicar respuestas.
Variaciones (escenarios secundarios)	
Excepciones	
Notas	

Tabla 20. Caso de uso detallado Responder a una pregunta con la API de Wikipedia.

6.5.3 Responder a una pregunta con NHS

Responder a una pregunta con NHS	
Precondiciones	El sistema debe tener el texto de la pregunta y el id del *tweet* en el que se realiza, y la categoría predicha en el Caso de uso 1 debe ser Salud.
Poscondiciones	El ultimo id respondido se actualiza al del *tweet* que acabamos de responder y el usuario recibe un *tweet* de respuesta.
Actores	La interfaz de Twitter, el usuario de Twitter, el modelo de clasificación y la API de Google con el motor de búsqueda personalizado de NHS.
Descripción	El sistema utilizará el texto de la pregunta y la API para obtener una respuesta, guardará dicha información y el enlace correspondiente de NHS, después mediante Twitter*Answer*er responderá al usuario y almacenará el id del *tweet* respondido para asegurarse de no duplicar respuestas.
Variaciones (escenarios	

secundarios)	
Excepciones	
Notas	

Tabla 42. Responder a una pregunta con NHS

6.5.4 Responder a una pregunta con la API de Google

Responder a una pregunta con la API de Wikipedia	
Precondiciones	El sistema debe tener el texto de la pregunta y el id del *tweet* en el que se realiza, y la categoría predicha en el Caso de uso 1 debe ser cualquiera de las no mencionadas anteriormente.
Poscondiciones	El ultimo id respondido se actualiza al del *tweet* que acabamos de responder y el usuario recibe un *tweet* de respuesta.
Actores	La interfaz de Twitter, el usuario de Twitter, el modelo de clasificación y la API de Google.
Descripción	El sistema utilizará el texto de la pregunta y la API para obtener una respuesta, guardará dicha información y el enlace correspondiente, que puede ser de cualquier página, después mediante Twitter*Answer*er responderá al usuario y almacenará el id del *tweet* respondido para asegurarse de no duplicar respuestas.
Variaciones (escenarios secundarios)	
Excepciones	
Notas	

Tabla 43. Responder a una pregunta con la API de Google.

6.5.5 Entrenar un modelo

Entrenar un modelo	
Precondiciones	Debemos tener un *dataset* que nos sirva para clasificar texto por categoría.
Poscondiciones	Se almacena el vectorizador del texto y el modelo entrenado.

Actores	El *dataset*, la biblioteca de Scikit-learn y el administrador.
Descripción	El administrador una vez haya encontrado un *dataset* acorde a las características que buscamos, deberá preprocesar el texto con el fin de normalizarlo, después creará un vectorizador para facilitar el aprendizaje del modelo, después entrenará el modelo de acuerdo con los diferentes algoritmos de *machine learning* y decidirá en función de su acierto cual será mejor para almacenar.
Variaciones (escenarios secundarios)	Cada algoritmo requerirá una ejecución distinta, además se harán pruebas con el preprocesamiento y modificaciones en los parámetros de cada algoritmo para obtener el resultado más certero posible.
Excepciones	
Notas	

Tabla 44. Entrenar un modelo.

6.6 Relación Escenarios – Casos de Uso – Requisitos

Aquí se muestra la relación entre los requisitos y los casos de uso:

Requisitos	**Realizar una pregunta**	**Responder con la API de Wikipedia**	**Responder con NHS**	**Responder con la API de Google**
RTW.1	X			
RTW.2	X			
RTW.3	X			
RTW.4	X			
RPR.1	X			
RPR.2	X			
RPR.3	X			
RCLF.1	X			
RCLF.1.1	X			
RCLF.1.2	X			
RCLF.1.3	X			
RCLF.1.4	X			
RCLF.1.5	X			
RCLF.1.6	X			
RCLF.1.7	X			
RCLF.1.8	X			
RCLF.1.9	X			
RCLF.1.10	X			
RB.1		X	X	X
RB.1.1		X		
RB.1.2			X	X
RB.1.3				
RB.2		X	X	X
RB.3		X		
RB.4			X	
RB.5				X

Tabla 45. Relación entre requisitos y casos de uso.

6.7 Análisis de Interfaces de Usuario

En este proyecto, no ha sido necesaria la realización de interfaces de usuario, ya que la totalidad de la experiencia de éste es en Twitter, la única decisión que se ha tomado en ese sentido es, en el *tweet* de respuesta al usuario incluir un enlace por si el usuario desea más información.

6.8 Especificación del Plan de Pruebas

En esta sección se creará y diseñará el plan de pruebas de la aplicación y sus funciones, así como todos los mecanismos que utilizaremos para detectar errores y corregirlos ya en la fase de implementación.

6.8.1 Pruebas unitarias

A lo largo del desarrollo del proyecto se realizarán pruebas unitarias para cada módulo creado, el mayor reto en este sentido serán las pruebas que se realizarán durante la creación del modelo clasificador, ya que el acierto de este será lo más influyente en el correcto funcionamiento del sistema.

6.8.2 Pruebas de Sistema e Integración

Debido a la naturaleza de nuestro sistema, se crearán diferentes pruebas que, de forma manual prueben la comunicación entre nuestros subsistemas una vez estos se hayan terminado.

6.8.3 Pruebas de usabilidad y rendimiento

Dado que no se tiene una interfaz de usuario no será necesario realizar pruebas de usabilidad.

Para el rendimiento se comprobará de que el usuario recibe una respuesta a su *tweet* en un tiempo adecuado, así como la evaluación del tiempo tardado en generar los modelos realizada previamente en el apartado *Evaluación y comparativa de resultados*.

Capítulo 7. Diseño del Sistema

En este capítulo se explicará, mediante diagramas, el diseño final que tiene nuestro sistema.

7.1 Arquitectura del Sistema

En primer lugar, se muestran los diagramas referentes a la arquitectura.

7.1.1 Diagramas de Paquetes

Los paquetes de nuestro sistema se organizan de la siguiente manera:

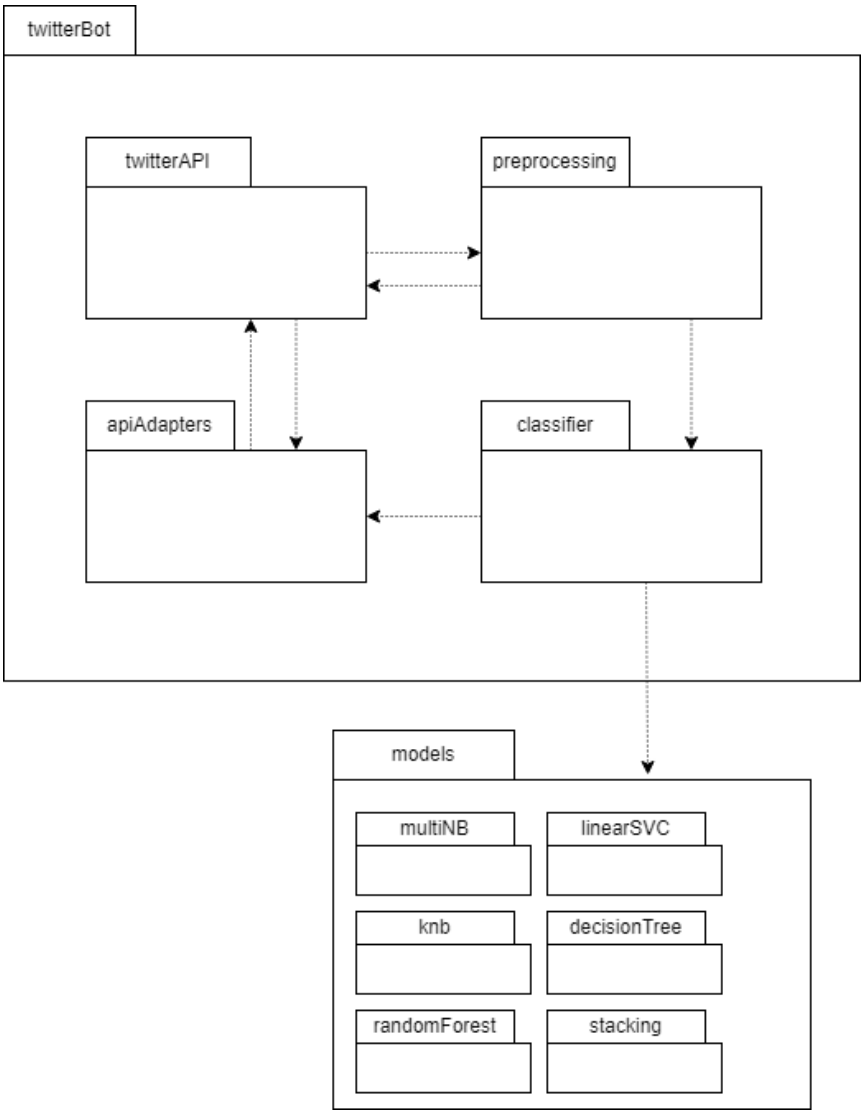

Imagen 27. Diagrama de Paquetes.

7.1.1.1 TwitterApi

Este paquete contiene todo lo relacionado con la API de Twitter, se encuentra la clase responsable de realizar la comunicación mediante tweepy, la encargada de buscar el siguiente *tweet* no respondido, y la que publica los *tweet*s de respuesta.

7.1.1.2 PreProcessing

En este paquete se encuentra todo lo relacionado con el preprocesamiento, empezando por los pasos indivi*dual*es hasta la clase PreProcessingHandler que decide qué pasos se seguirán y es la que se utiliza como enlace a los demás paquetes.

7.1.1.3 Classifier

Este paquete es el encargado de la clasificación de las preguntas, se encarga de cargar el modelo clasificador y realizar la predicción. Una vez realizada se envía al paquete api*Adapter*s.

7.1.1.4 Models

Este paquete guarda los modelos entrenados para la clasificación del texto.

7.1.1.5 ApiAdapters

En este paquete se encuentra la clase *ApiHandler*, que decide que API debemos utilizar y hace de enlace con los demás paquetes, así como las clases que adaptan las APIs que vamos a utilizar a nuestro *ApiHandler*.

7.1.2 Diagramas de Componentes

Se muestra a continuación el diagrama de componentes de nuestro sistema:

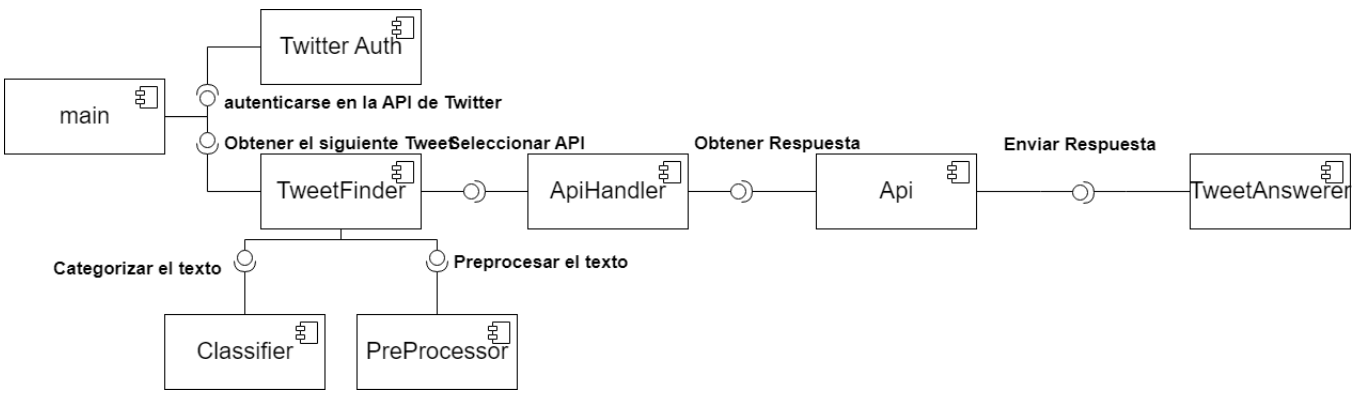

Imagen 28. Diagrama de componentes.

7.1.2.1 Main

El encargado de arrancar el sistema se autentifica mediante *TwitterAuth* y llama al método de *TweetFinder* para buscar el siguiente *tweet*.

7.1.2.2 TwitterAuth

Este componente se encarga de iniciar la conexión con la API de Twitter.

7.1.2.3 TweetFinder

Componente que se encarga de comprobar si han llegado *tweet*s nuevos, si han sido respondidos y si no, enviarlos a los demás componentes para proceder con la clasificación.

7.1.2.4 PreProcessing

Este componente recibe el texto crudo y lo trata de la misma forma de la que se trató el modelo entrenado.

7.1.2.5 Classifier

El componente *Classifier* recibe el texto después de pasar por el preprocesamiento, carga el modelo entrenado y realiza una predicción sobre su categoría.

7.1.2.6 ApiHandler

Este componente se encarga de decidir que API se utilizará para responder al usuario en función de la categoría predicha en *Classifier*.

7.1.2.7 Api

Componente que recibe la pregunta que se ha realizado, adapta la API elegida para a nuestro sistema y envía la respuesta resumida y el enlace para obtener más información al *TweetAnswerer*.

7.1.2.8 TweetAnswerer

Este último componente, una vez recibida la respuesta y el enlace, responde al *tweet* original con dicha información.

7.1.3 Diagramas de Despliegue

En cuanto al despliegue del sistema, se hará localmente, a excepción de la cuenta de Twitter del proyecto, por tanto, el diagrama de despliegue de nuestro sistema será el siguiente:

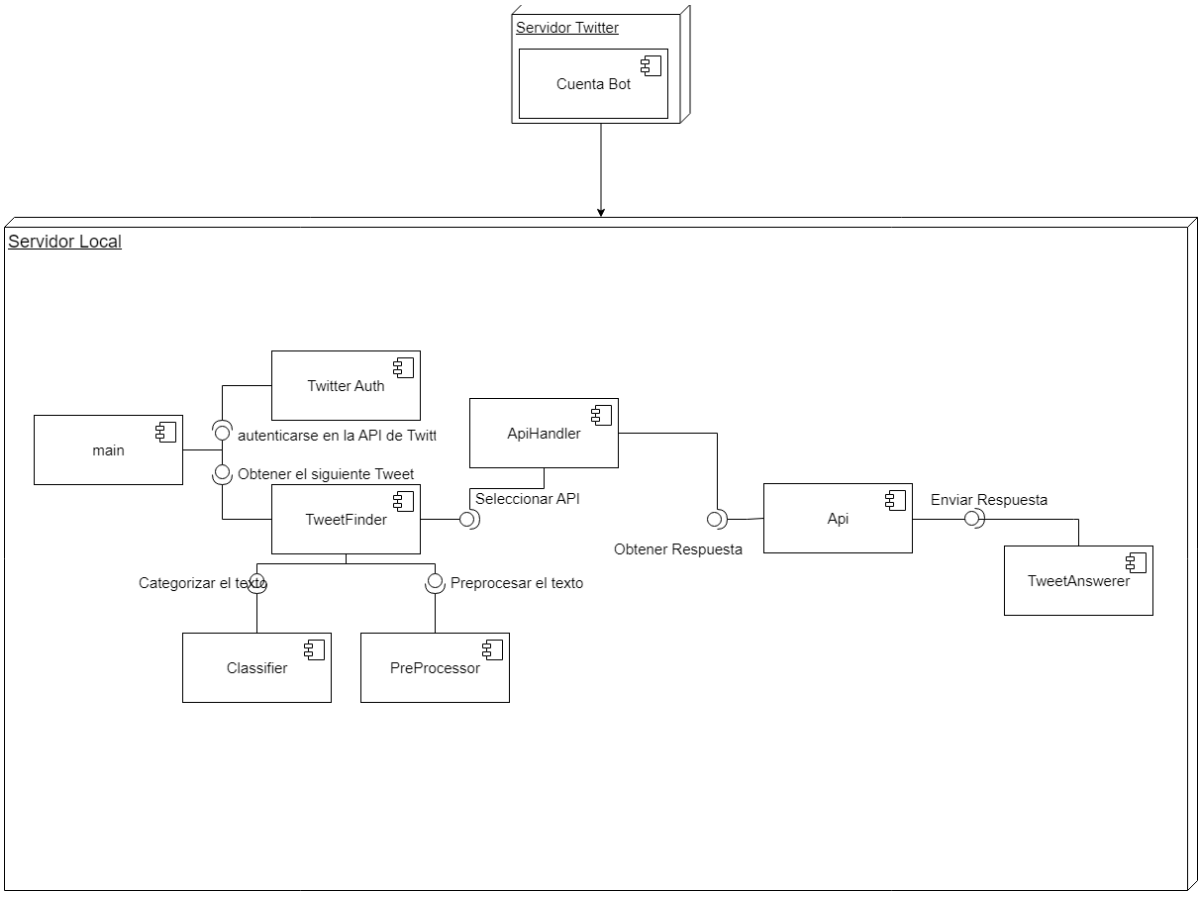

Imagen 29.Diagrama de Despliegue.

7.2 Diseño de Clases

A continuación, se muestra un diagrama de las clases del sistema:

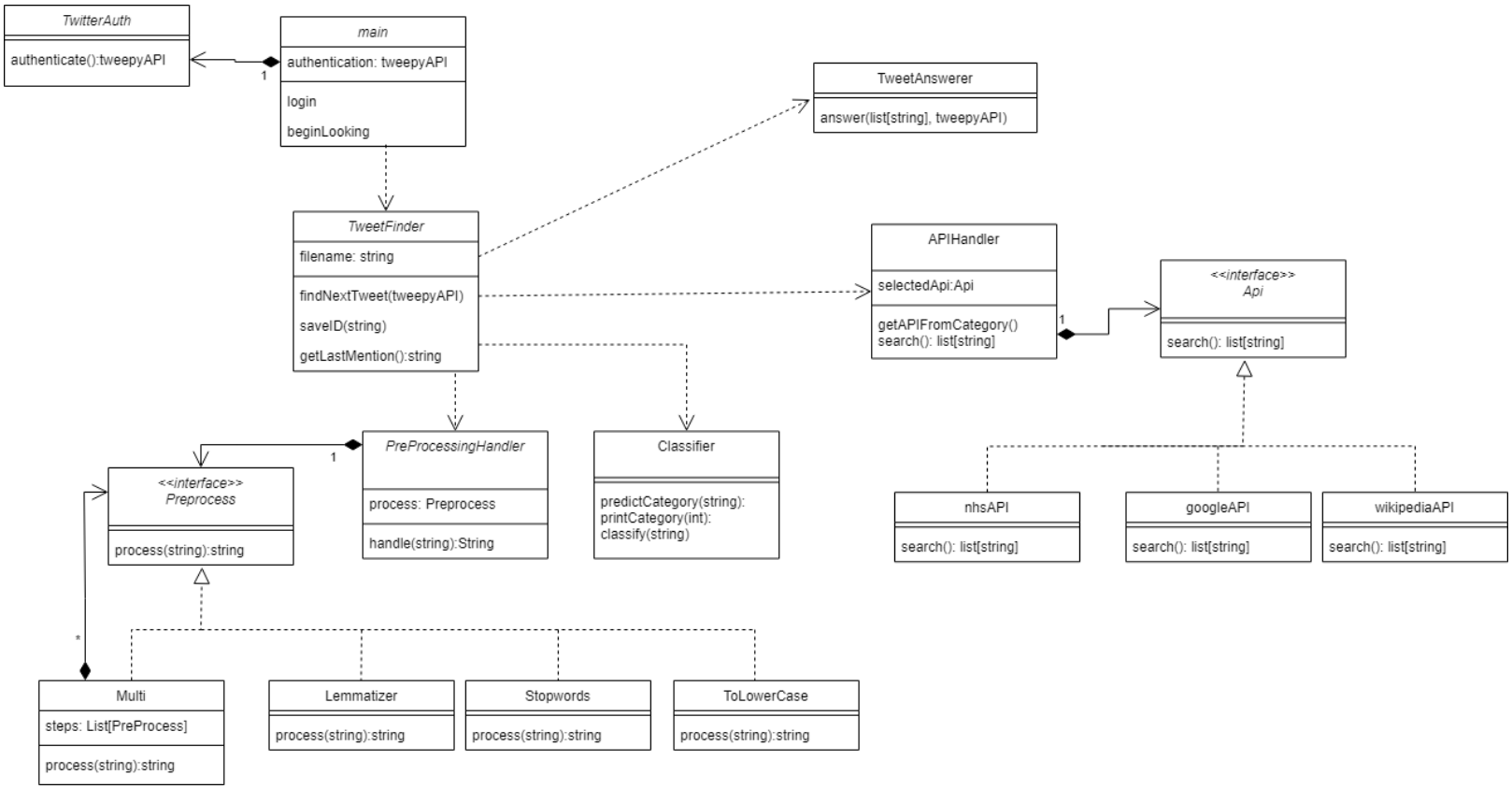

Imagen 30. Diagrama de clases.

En términos generales, se ha seguido el diseño explicado previamente en el apartado *Diagrama de Clases* del Análisis, con la inclusión de la clase *main*, que será la encargada del arranque del programa, y de las clases que se encargan de englobar el funcionamiento de los subsistemas y actuar de enlace entre ellos, como son *ApiHandler* y *PreProcessingHandler*.

En el apartado *Descripción Detallada de las Clases* se puede encontrar una explicación detallada de cada una de ellas.

7.3 Diagramas de Interacción y Estados

En esta sección se muestra el diagrama de interacción con todo el proceso que realiza nuestro sistema, y posteriormente, para su mejor visualización, se mostrarán por separado los diagramas de los dos casos de uso que lo componen. En lo referente a los casos de uso de responder mediante APIs se agruparán en una generalización ya que los estados no varían según la API escogida, para evitar repeticiones.

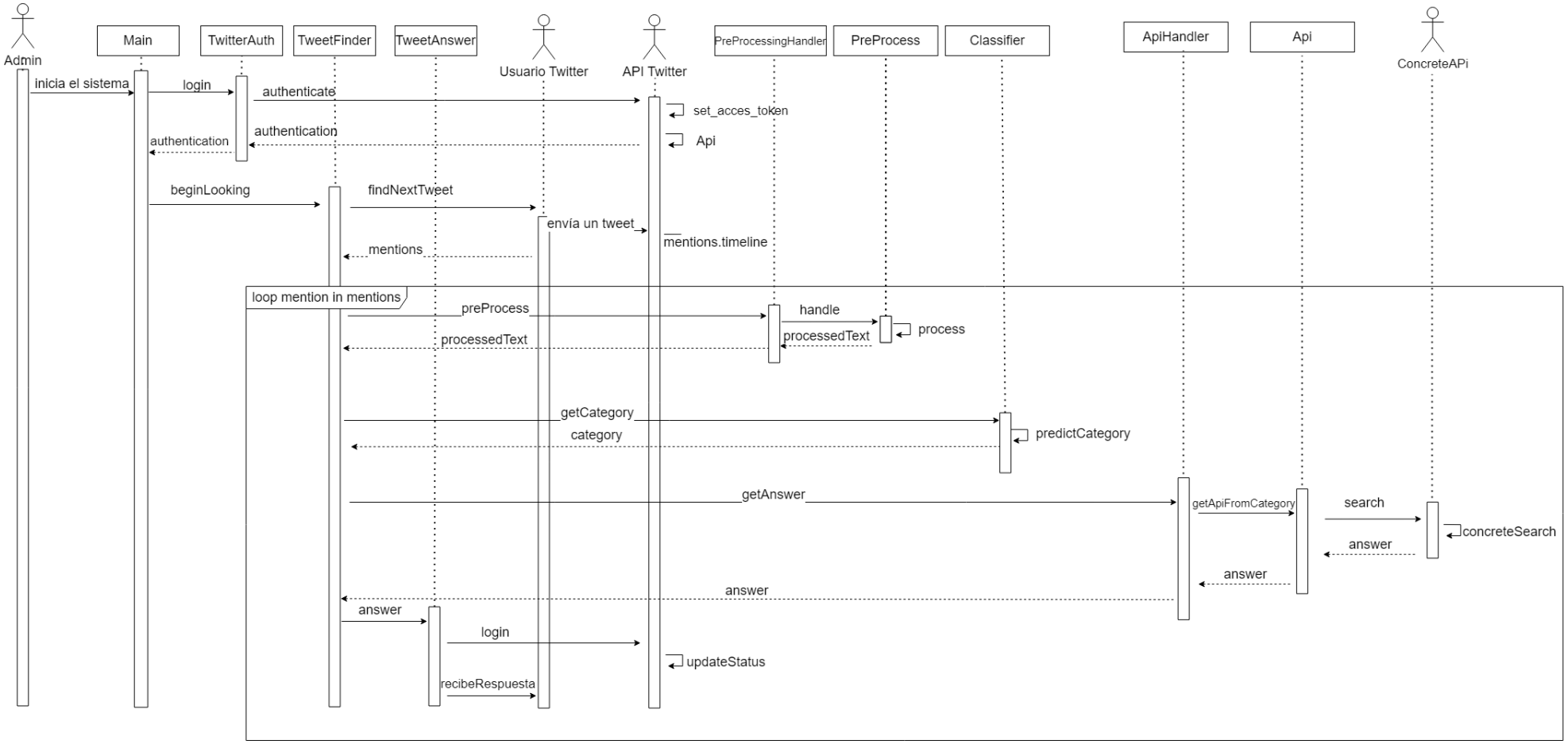

Imagen 31.Diagrama de estados del sistema.

7.3.1 Realizar una pregunta

Extrayendo del diagrama anterior tan solo lo perteneciente al caso 1 se obtiene el siguiente diagrama:

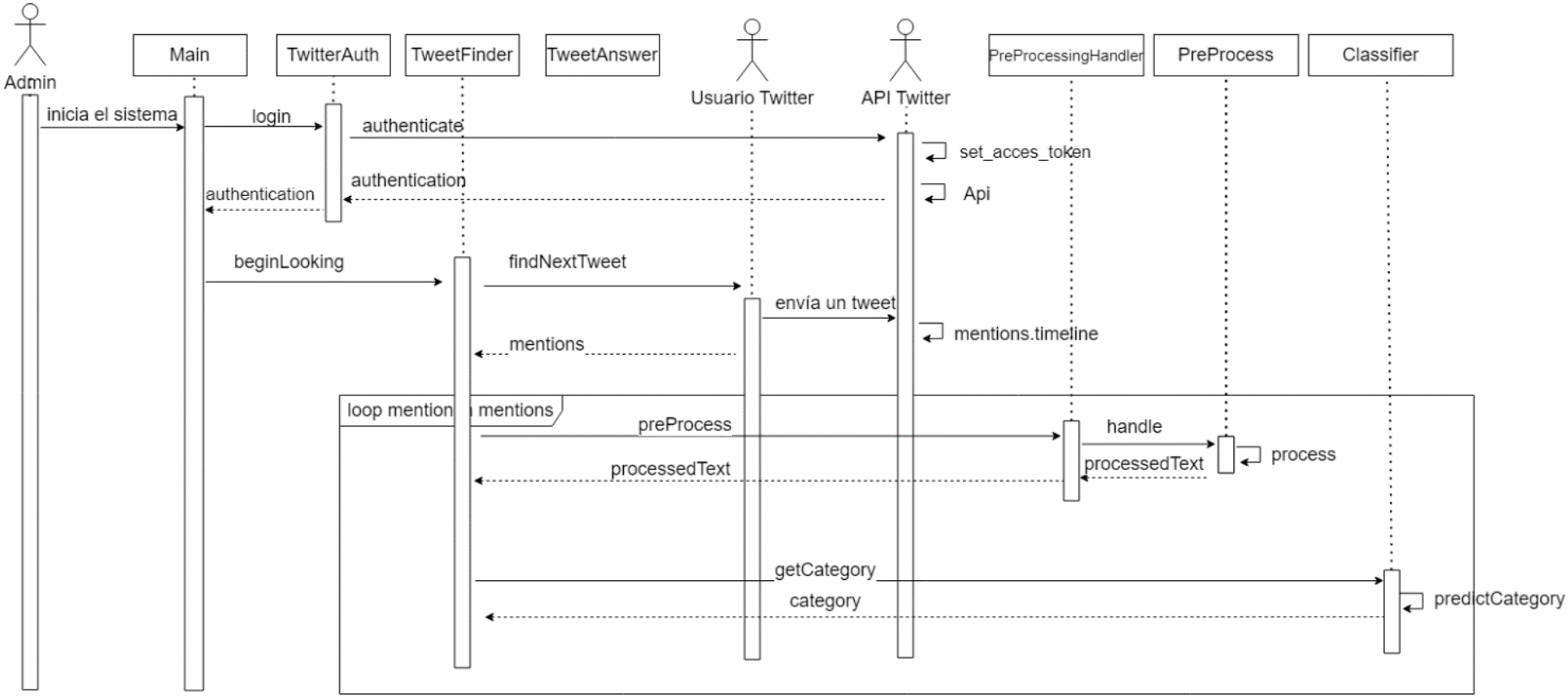

Imagen 32. Diagrama de estados para el Caso de uso 1: Realizar una pregunta.

7.3.2 Responder a una pregunta con una API

Para los casos 2, 3 y 4 el diagrama sería el mismo. La única modificación seria que en lugar de Api aparecería la clase encargada de adaptar la API concreta:

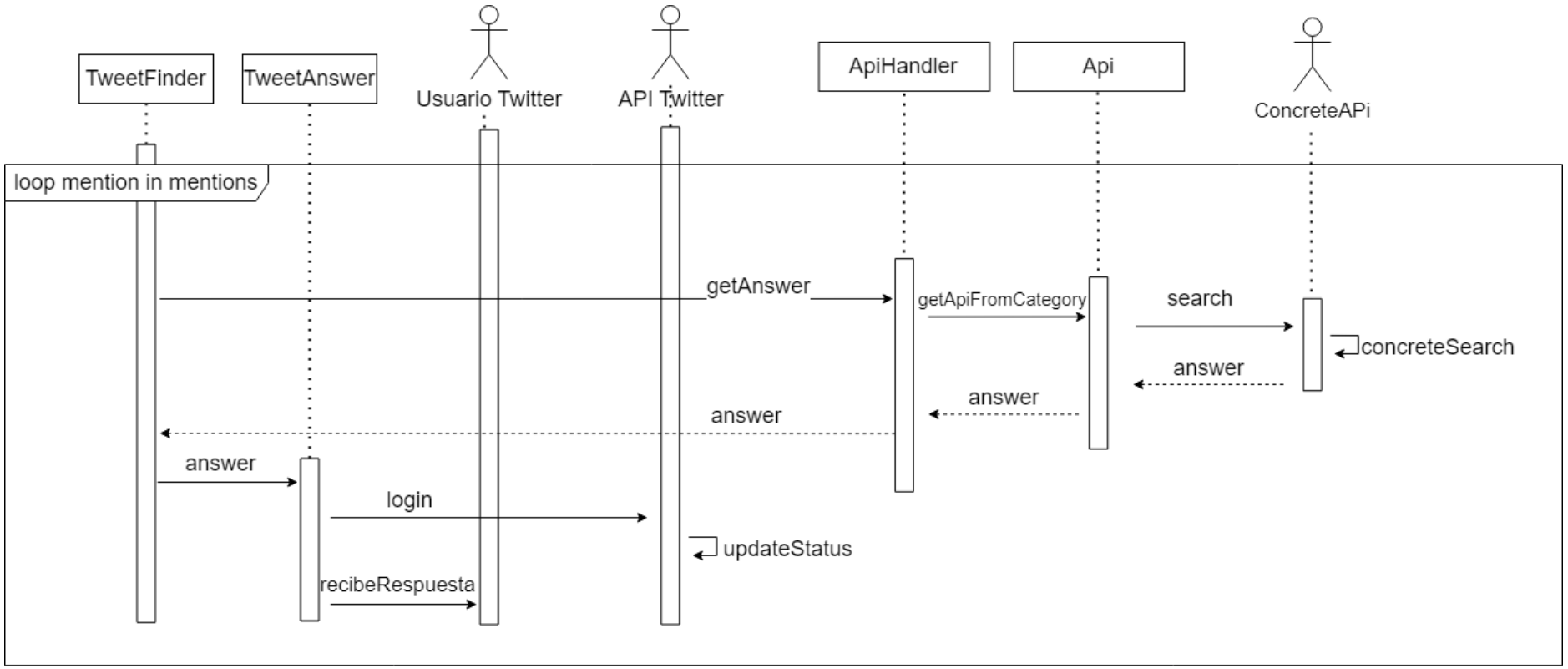

Imagen 33. Diagrama de estados Caso de uso Responder mediante API.

7.4 Diagramas de Actividades

A continuación, se muestra el diagrama de actividad de nuestro sistema:

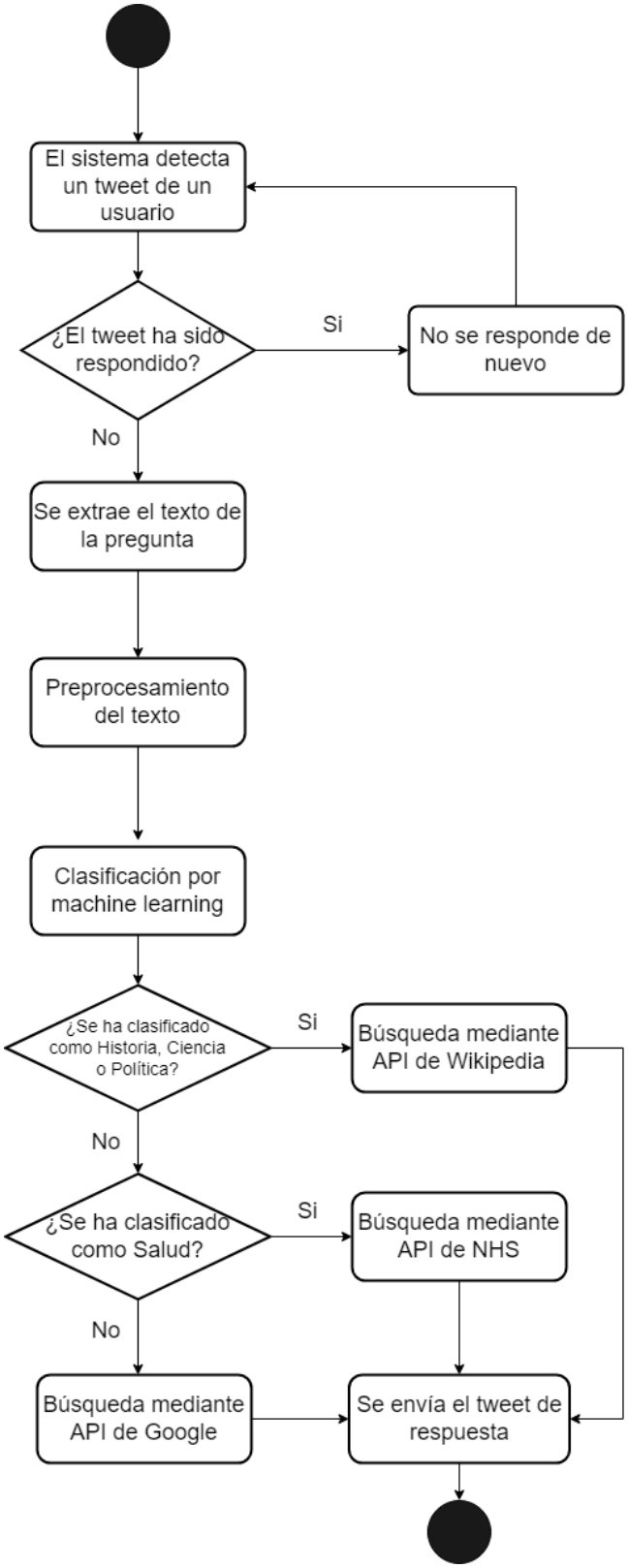

Imagen 34. Diagrama de Actividad.

7.5 Diseño de la Base de Datos

En nuestro sistema no ha sido necesario la creación de una base de datos, el único dato que se guarda de forma local es el id del último *tweet* respondido.

7.6 Diseño de la Interfaz

Dada la naturaleza de este proyecto, no es necesario el diseño de una interfaz, ya que la interacción con el usuario se realiza vía Twitter

7.7 Especificación Técnica del Plan de Pruebas

Durante el proceso de construcción de nuestro sistema se han realizado una serie de pruebas que iremos se detallan a continuación.

7.7.1 Pruebas Unitarias

Para comenzar, se dividirán las pruebas unitarias según los subsistemas de nuestro sistema.

7.7.1.1 Pruebas unitarias de la interfaz con Twitter

Primero se muestran las pruebas del apartado relacionado con la red social:

Responsable	Prueba	Resultado Esperado
TwitterAuth	Conexión mediante tweepy a la API de Twitter con las credenciales de nuestra cuenta	El sistema es capaz de conectarse a Twitter y nos devuelve el objeto del tipo tweepy.API con el que podremos trabajar.
TweetFinder	Recopilación de *tweet*s	Se recopilan todos los *tweet*s en los que se ha mencionado a nuestra cuenta.
TweetFinder	Distinción de *tweet*s respondidos	Sólo se devuelven los *tweet*s posteriores al último *tweet* respondido, guardado en el last_id, o en caso de que ese archivo no se encuentre, los *tweet*s posteriores al arrancado del sistema.
TweetFinder	*Tweet* nuevo de un usuario	Con el sistema ya arrancado, se detecta un nuevo *tweet* de un usuario.
TweetAnswerer	Responder a un *tweet*	El sistema es capaz de responder a un *tweet* de un usuario una sola vez.
TweetAnswerer	Guardar último id respondido	Cuando se ha respondido a un usuario se guarda el id del *tweet* al que se responde como last_id.

Tabla 46. Pruebas Unitarias Twitter.

7.7.1.2 Pruebas unitarias preprocesamiento

A continuación, las pruebas del apartado de preprocesamiento:

Responsable	Prueba	Resultado Esperado
Stopwords	Eliminación de palabras vacías	Dado un texto, se devuelve con las palabras vacías eliminadas del mismo.
toLowerCase	Conversión a minúscula	El texto se devuelve convertido a minúsculas.
Lemmatizer	Lematización	Las palabras del texto que aparecen en el diccionario de WordNet se convierten a lemas.
Multi	Combinaciones de pasos	Se pueden crear distintas combinaciones de los pasos y se ejecutan correctamente como uno solo.

Tabla 47. Pruebas Unitarias Preprocesamiento.

7.7.1.3 Pruebas unitarias del Clasificador

Con respecto al clasificador, dado que es un módulo de una sola clase se realizará un único tipo de prueba:

Responsable	Prueba	Resultado Esperado
Classifier	Categorizar una pregunta	Con un texto procesado, el clasificador devuelve una categoría relacionada con el texto.

Tabla 48. Pruebas Unitarias Clasificador.

Para la realización de esta prueba se creó el método *printCategory*, que devuelve en forma de texto la categoría a la que pertenece un texto, aunque en el uso normal del sistema no se vaya a usar, dada su utilidad para las pruebas o si se ampliase el proyecto, se mantuvo en la clase *Classifier*.

7.7.1.4 Pruebas unitarias del módulo de APIs

De la misma forma, se probará el correcto funcionamiento de cada clase que adapta una API:

Responsable	Prueba	Resultado Esperado
WikipediaAPI	Responder con la API de Wikipedia	Dado un texto, se devuelve una respuesta con el inicio del texto de la página de Wikipedia más cercana y su enlace.
NHSAPI	Responder con la API de NHS	Dado un texto, se devuelve una respuesta con el inicio del texto de la página de NHS más cercana y su enlace.
GoogleAPI	Responder con la API de Google	Dado un texto, se devuelve una respuesta con el inicio del texto del primer resultado que ofrece la API de Google *Search* y su enlace.

Tabla 49. Pruebas Unitarias APIs.

7.7.2 Pruebas de Integración y del Sistema

Las pruebas de integración introducidas en el apartado *Pruebas de Sistema e Integración* que se realizarán serán las siguientes:

Prueba	Resultado Esperado
Preprocesar el contenido de un *tweet*	El sistema es capaz enviar el texto de un *tweet* a su preprocesamiento y devolver debidamente procesado.
Clasificar un *tweet* como Salud	Teniendo un *tweet* que hace referencia a un problema de salud, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Salud.
Clasificar un *tweet* como Política	Teniendo un *tweet* que hace referencia a un tema político, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Política.
Clasificar un *tweet* como Deportes	Teniendo un *tweet* que hace referencia a deportes, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Deportes.
Clasificar un *tweet* como Finanzas	Teniendo un *tweet* que hace referencia negocios o finanzas, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Finanzas.
Clasificar un *tweet* como Historia	Teniendo un *tweet* que hace referencia a un hecho histórico, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Historia.
Clasificar un *tweet* como Ciencia	Teniendo un *tweet* que hace referencia a un hecho científico, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Ciencia.
Clasificar un *tweet* como Familia	Teniendo un *tweet* que hace referencia a un problema familiar, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Familia.
Clasificar un *tweet* como Educación	Teniendo un *tweet* que hace referencia a un tema educativo, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Educación.
Clasificar un *tweet* como Informática	Teniendo un *tweet* que hace referencia a la informática, una vez preprocesado el texto debe llegar al clasificador, que nos

	devolverá como categoría Informática.
Clasificar un *tweet* como Entretenimiento	Teniendo un *tweet* que hace referencia al entretenimiento, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Entretenimiento.
Clasificar un *tweet* como Sociedad	Teniendo un *tweet* que hace referencia a un tema de sociedad, una vez preprocesado el texto debe llegar al clasificador, que nos devolverá como categoría Sociedad.
Responder a un *tweet* de Salud	Cuando se haya categorizado un *tweet* como Salud, el sistema debe ser capaz de enviar un *tweet* de respuesta y responder con información de la página de NHS.
Responder a un *tweet* de Historia, Ciencia o Política	Cuando se haya categorizado un *tweet* como Salud, el sistema debe ser capaz de enviar un *tweet* de respuesta y responder con información de Wikipedia.
Responder a un *tweet* de otra temática	Cuando el *tweet* no pertenezca a las temáticas anteriores, el sistema debe ser capaz de enviar un *tweet* de respuesta y responder con información de Wikipedia.

Tabla 50. Pruebas de Integración.

7.7.3 Pruebas de Usabilidad y Accesibilidad

Dada la carencia de interfaz propia de nuestro sistema, la usabilidad y accesibilidad dependerán de Twitter.

Capítulo 8. Implementación del Sistema

En este capítulo se especificarán los detalles sobre los estándares, lenguajes y herramientas utilizados en el desarrollo del sistema.

8.1 Estilos y normas

Para el desarrollo en Python del sistema se trató de seguir PEP-8 [58], siglas de Python Enhancement Proposal, realizada por Guido Van Rossum, el diseñador del lenguaje.

8.2 Lenguajes de programación

El sistema está en su totalidad desarrollado mediante un único lenguaje de programación.

8.2.1 Python

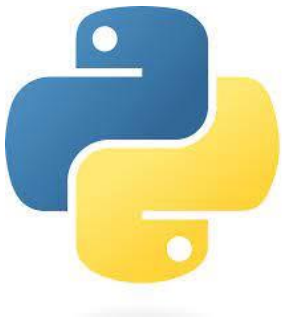

Imagen 35. Logo de Python.

Python [59] es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y la programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma[60].

Si bien Python estaba lejos de ser el lenguaje del que más conocimientos se tenía, al ser el más extendido en *machine learning* fue el elegido para el desarrollo. Además, Python es un lenguaje que intenta, y consigue, ser altamente entendible si se tiene experiencia previa trabajando con otros lenguajes [61].

8.3 Herramientas y programas usados

A continuación, se describirán de los programas utilizados en el desarrollo del sistema.

8.3.1 PyScripter

Imagen 36. Logo PyScripter.

PyScripter [62] es un entorno de desarrollo integrado (IDE) gratuito y de código abierto. Originalmente comenzó como un IDE ligero diseñado para servir al propósito de proporcionar una solución de scripting para aplicaciones Delphi. Con el tiempo, evolucionó hasta convertirse en un IDE independiente de Python con todas las funciones [63].

Es un entorno sencillo y ligero que ya se había utilizado anteriormente de forma breve durante el grado, por lo que se usó sobre todo en las etapas iniciales de aprendizaje.

8.3.2 PyCharm

Imagen 3738. Logo PyCharm.

PyCharm [64] es un entorno de desarrollo integrado (IDE) de Python que proporciona una amplia gama de herramientas esenciales para los desarrolladores de Python, estrechamente integradas para crear un entorno conveniente para el desarrollo productivo de ciencia de datos, web y Python [65].

Una vez familiarizados con Python, el entorno de PyScripter empezaba a carecer de funcionalidades como el poder crear un proyecto entero dentro del entorno, en lugar de sólo scripts indivi*dual*es, avisos si no se seguía la guía de estilo, etc.

8.3.3 Microsoft Project

Imagen 3940. Logo Microsoft Project.

Microsoft Project (o MSP) [66] es un software de administración de proyectos y programas de proyectos desarrollado y comercializado por Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo [67].

Utilizado, en su versión de 2019, principalmente para la planificación y cálculo de presupuestos del proyecto.

8.3.4 Opera GX

Imagen 41. 42. Logo Opera GX.

Opera [68] es un navegador web creado por la empresa noruega Opera Software. Actualmente propiedad de Golden Brick Capital, empresa China de inversión. Usa el motor de renderizado Blink. Tiene versiones para computadoras de escritorio, teléfonos móviles y tabletas. La versión GX añade funcionalidades cómo la posibilidad de limitar el uso que realiza el navegador de la memoria RAM y la CPU [68], [69].

En este proyecto, se utilizó para el proceso de investigación, así como para comprobar el correcto funcionamiento del Bot.

8.3.5 Twitter

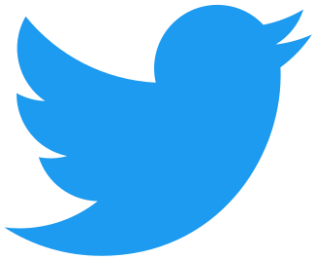

Imagen 43. 44. Logo Twitter.

Twitter [70] es un servicio de micro blogueo, permite enviar mensajes de texto plano de corta longitud, con un máximo de 280 caracteres (originalmente 140), que se muestran en la página principal del usuario. Los usuarios pueden suscribirse a los *tweet*s de otros usuarios. Por defecto, los mensajes son públicos, pudiendo difundirse privadamente mostrándose únicamente a unos seguidores determinados [71].

En nuestro caso, realiza las funciones de interfaz mediante la cual el usuario puede comunicarse con el sistema.

8.3.6 Putty

Imagen 45. Logo Putty.

Putty [72] es un emulador de terminal gratuito y de código abierto. Soporta varios protocolos de red, incluyendo SCP, SSH, Telnet, *rlogin*, y conexión *raw socket*. También puede conectarse a un puerto serie. El nombre " Putty " no tiene significado oficial [73].

Inicialmente, cuando se empezaron a entrenar los modelos en una máquina de la Universidad, se utilizó Putty para realizar la conexión con ésta.

8.3.7 WinSCP

Imagen 46. Logo WinSCP.

WinSCP [74] es una aplicación libre y de código abierto. WinSCP es un cliente SFTP gráfico para Windows que emplea SSH. El anterior protocolo SCP también puede ser empleado. Su función principal es facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios SSHNewbie [75].

Se utilizó para transferir los archivos entre nuestra máquina local y la proveída por la Universidad.

8.3.8 MobaXTerm

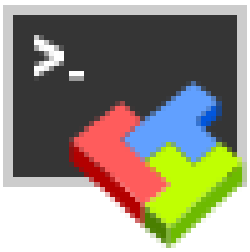

Imagen 47. Logo MobaXTerm.

MobaXTerm [76] es una aplicación que proporciona capacidad X-Server para el sistema operativo Microsoft Windows. Esto permite que las aplicaciones que se ejecutan en el entorno Unix/Linux muestren interfaces gráficas de usuario en el escritorio de Microsoft Windows [77].

Se utilizó posteriormente a Putty y WinSCP ya que combina las funcionalidades de ambos además de tener integrado un editor de texto, lo que facilita el desarrollo.

8.4 Módulos y librerías utilizados.

En esta sección se entrará en detalle sobre los módulos y las librerías que se han utilizado durante el desarrollo.

8.4.1 Scikit-Learn

Imagen 48. Logo Scikit-Learn.

Scikit-learn [42] es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye varios algoritmos de clasificación, regresión y análisis de grupos entre los cuales están máquinas de vectores de soporte, Random Forests, Gradient boosting, *K-means* y *DBSCAN* [78].

Se ha utilizado para crear y entrenar los modelos de *machine learning* capaces de categorizar las preguntas de los usuarios.

8.4.2 NLTK

Imagen 49. Logo NLTK.

El kit de herramientas de lenguaje natural, o más comúnmente conocido como NLTK [79], es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural simbólico y estadísticos para el lenguaje de programación Python. NLTK incluye demostraciones gráficas y datos de muestra [80].

El preprocesamiento en este proyecto se realiza gracias a esta herramienta.

8.4.3 Tweepy

Tweepy [81]es una biblioteca de Python para integrarse con la API de Twitter. Debido a que Tweepy está conectado con la API de Twitter, puede realizar consultas complejas además de recopilar *tweet*s. Le permite aprovechar todas las capacidades de la API de Twitter [82].

En este sistema es esencial para poder realizar acciones en nuestra cuenta de Twitter.

8.4.4 Pickle

El módulo pickle implementa protocolos binarios para serializar y deserializar una estructura de objetos Python. *Pickling* es el proceso mediante el cual una jerarquía de objetos Python se convierte en un flujo de bytes mientras que *unpickling* es la operación inversa, mediante la cual un flujo de bytes (de un archivo binario o un objeto tipo bytes) se convierte de nuevo en una jerarquía de objetos [83].

En el sistema sirve para poder almacenar un modelo tras ser entrenado y cargarlo posteriormente cuando vaya a ser utilizado.

8.4.5 MediaWiki API

Imagen 50. Logo MediaWiki.

MediaWiki [84] es un software para wikis libre, programado en el lenguaje PHP y utilizado por Wikipedia. Su API nos permite funcionalidades como la autenticación, las operaciones en páginas y la búsqueda en Wikipedia [85].

En nuestro caso es utilizado para buscar respuestas a ciertos tipos de preguntas, buscando el artículo de Wikipedia realtivo a la pregunta.

8.4.6 Google Programmable *Search* Engine

Imagen 51. Logo Google Programmable Search Engine.

El motor de búsqueda programable de Google [86] es una plataforma proporcionada por Google que permite a los desarrolladores web presentar información especializada en búsquedas web, refinar y categorizar consultas y crear motores de búsqueda personalizados, basados en la Búsqueda de Google [87].

En nuestro sistema se utiliza para hacer búsquedas en páginas como NHS, que si bien si contienen una gran variedad de información sobre un tema que resulta de utilidad (salud en este caso) no disponen de una API propia gratuita.

8.4.7 Pandas

Imagen 52. Logo Pandas.

Pandas [88] es una biblioteca de software escrita para el lenguaje de programación Python para la manipulación y el análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series de tiempo [89].

Es utilizado para cargar en memoria el *dataset* y que lo pueda utilizar el modelo para ser entrenado.

8.5 Problemas encontrados

A continuación, se detallan los problemas encontrados durante el desarrollo.

8.5.1 Problemas con la API de Twitter

Los primeros problemas encontrados estuvieron relacionados con la API de Twitter, a la hora de recuperar las menciones, el método que nos ofrecía Tweepy devolvía todas las menciones respondidas o no, por lo que se debía encontrar la forma de saber a qué *tweet*s se había respondido previamente. Además, estos debían guardarse aunque el sistema dejase de ejecutarse, ya que si no al arrancarlo de nuevo se volvería a responder a todos los *tweet*s.

La primera solución por la que se optó era guardar el id de los *tweet*s a los que se respondía. Inicialmente era una solución válida, pero conforme más *tweet*s tenía la aplicación, más tiempo tardaba en realizar la comprobación, ya que se tenían que comprobar todas las menciones con todos los ids guardados cada vez que se realizaba una iteración.

Después, se modificó para que se dejase de comprobar una vez encontrado el primer *tweet* respondido, el cual, por la forma en la que almacenan, debía ser siempre el último guardado, por lo que no era necesario guardar nada más que el último id respondido.

8.5.2 Problemas con los vectorizadores

Con respecto a los vectorizadores surgen dos problemas diferentes.

8.5.2.1 Tamaño de los vectorizadores en RAM

Durante el entrenamiento de los modelos se investigó sobre los diferentes vectorizadores, los cuales se trataron de generar para observar las diferencias en los resultados. Es en este proceso cuando empezaron a surgir errores en la máquina Windows sobre la que se estaba trabajando, ya que durante la ejecución de los programas que generaban los vectorizadores, el sistema se paralizaba por completo y al cabo de un tiempo saltaba en el intérprete el error de que no había suficiente memoria disponible para almacenar el vectorizador.

Para solucionar esto inicialmente se limitó el número de *features* del vectorizador, lo que reducía su tamaño. Sin embargo, tras realizar una investigación, se apreció que con el tamaño del *dataset* y la RAM disponible no debería surgir un error como el que estaba ocurriendo, así que, se solicitó una máquina a la Universidad para trabajar en ella y ver si ahí surgía también el error. Una vez en esa máquina, los vectorizadores se generaban de forma normal sin ningún problema de memoria.

8.5.2.2 Modelos con Recall y F1-score bajos

Durante la parte del desarrollo en la cual se investigaba sobre qué modelos darían los mejores resultados y se realizaban pruebas con los distintos algoritmos, aparecían habitualmente los siguientes resultados:

Splitter best Criterio: gini Masx Features: log2

	precision	recall	f1-score	support
1	0.82	0.12	0.21	28193
2	0.54	0.00	0.01	28068
3	0.11	0.98	0.20	28041
4	0.54	0.05	0.08	27980
5	0.63	0.09	0.16	27938
6	0.84	0.11	0.19	28075
7	0.83	0.02	0.03	27620
8	0.21	0.05	0.09	28002
9	0.70	0.06	0.11	28072
10	0.62	0.11	0.19	28011
accuracy			0.16	280000
macro avg	0.58	0.16	0.13	280000
weighted avg	0.58	0.16	0.13	280000

Imagen 53. Problemas con el Recall.

Si tan solo nos fijamos en la precisión, los resultados pueden parecer buenos ya que es del 60%, e inicialmente no reparamos en el problema que reflejan.

Es cuando se empieza a probar manualmente el modelo cuando nos damos cuenta de que a la hora de realizar una predicción el modelo nos devolvía siempre el valor 3, correspondiente a la categoría de Salud. En los resultados se observa que la precisión (positivos acertados dividido entre positivos predichos) es muy baja y el *Recall* (positivos acertados entre negativos fallados) es prácticamente del 100%. Esto se debe a que el modelo consideraba prácticamente la totalidad de los datos como esa categoría, y tan sólo categorizaba distintos aquellos con los que estaba claramente seguro de que pertenecían a otra, haciendo que en el resto de categorías la precisión fuese muy alta, y que a la hora de realizar la media de la precisión se equilibrase.

Cuando finalmente se reparó en el error, se valoró la posibilidad de que sea un problema del *dataset*, que, por algún motivo, tenga mal categorizadas las preguntas de salud o bien, que la considere muy amplia y que el resto también se pueden clasificar como tal. Como prueba, se eliminó dicha categoría del *dataset* para observar si los resultados mejoraban, pero el modelo devolvía unos datos similares, retornando una categoría diferente ahora como la más probable.

Se probó entonces a modificar el vectorizador, ya que era la otra posibilidad donde se podía encontrar el error. Previamente, la vectorización se daba en dos pasos, primero utilizábamos *CountVectorizer* y posteriormente *TfidfTransformer* [90]. En lugar de ello, para solucionar el error, se sustituyeron ambos por un *TfidfVectorizer*, observando que los resultados parecían arreglarse.

8.5.3 Problema con la API de NHS

Durante la fase de investigación en la que se pretendía encontrar APIs que se ajustasen a nuestras necesidades, se encontró que la página de NHS ofrece una gran variedad de posibilidades, entre ellas una API que trabaja con una sección de su portal que se dedica a responder a preguntas típicas de salud [91].

Imagen 54. Portal del desarrollador de NHS.

Viendo esto, se realizó el registro en el portal de desarrollador de la web, consiguiendo una Key de acceso.

Una vez obtenidas las credenciales, se observa la documentación, apreciando que es prácticamente inexistente. En cambio, en la otras que ofrecen, la documentación es más extendida. Cuando por fin se consigue la conexión, utilizando en parte la documentación del resto de APIs, se advierte que realmente la API no parece estar desarrollada ya que ante cualquier petición correcta devuelve el código 200 y una respuesta vacía.

Imagen 55. Respuesta API NHS.

8.6 Descripción Detallada de las Clases

Se describirá ahora en detalle cada clase y los métodos que contiene, al igual que se ha realizado previamente, dividiendo cada clase según su subsistema.

8.6.1 Interfaz de Twitter

En este apartado, como ya se ha descrito se encuentran las siguientes clases.

8.6.1.1 TweetFinder

TweetFinder es la clase central de nuestro sistema, ya que el bucle por el cual se comprueba la llegada de nuevos *tweet*s se encuentra en ella. En ella se encuentran tres métodos:

GetLastMentionID es el método encargado de cargar el archivo en el que se ha guardado el id del último *tweet* al que se ha respondido y devolver su valor. En el caso en el que no se encuentre el archivo o este esté vacío devolverá -1.

SaveID recibe como parámetro el id del último *tweet* que se ha respondido y actualiza el archivo de texto con este id.

FindNextTweet es un método que recibe como parámetro la autenticación realizada en la clase *TwitterAuth*. Primero llama al método *GetLastMentionID*, después recorre mediante un bucle la lista de *tweet*s que devuelve el método de tweepy *mentions_timeline* y en ese bucle realiza lo siguiente:

1. Inicia una variable *found* (encontrado) como *false* que será la encargada de saber si se ha encontrado un *tweet* al que ya se ha respondido.
2. Comprueba que la última id respondida no sea -1, de ser así, comprobará si la hora de publicación de un *tweet* es posterior a la de arranque del sistema, y si no se da este caso, la variable *found* será puesta como verdadera.
3. En caso de que sí exista un último id, lo compara con el id de la mención sobre el que estamos iterando. Si es así, la variable *found* se pone como verdadera.
4. Llegado a este punto, si la variable *found* es verdadera se para el bucle ya que se ha llegado a un *tweet* ya respondido.
5. En cambio, si no es verdadera quiere decir que se debe responder al *tweet,* por lo que se llama al preprocesamiento, clasificación y posterior búsqueda de respuesta del texto. Después, se llama al método *answer* de *TweetAnswerer* con la respuesta obtenida y el id del *tweet* al que responder. Por último, se almacena el id del *tweet* respondido en una lista, y se declara una variable *answered* a true.

SI la variable *answered* es verdadera llegados a este punto en la ejecución, se llama al método *saveID* con el primer valor de la lista de ids a los que se ha respondido, ya que se responden en orden de más recientes primero, por lo que el primer id será el del *tweet* más reciente.

Después se llama al método *time.sleep*, introduciendo el valor 30 como parámetro, para que el sistema solo llame a este método cada 30 segundos, ya que se ha decido que sea el período de tiempo que esperará nuestro sistema para buscar nuevas menciones.

8.6.1.2 TwitterAuth

TwitterAuth es una clase que se encarga de la autenticación en Twitter mediante tweepy, con un único método.

Authenticate es un método con 4 variables locales, *apiKey* y *apiKey_secret*, que se utilizan como parámetros en la llamada al método *tweepy.OAuthHandler,* que devuelve un objeto *auth*, con el cual podremos iniciar nuestra conexión. Primero se debe incluir el token de acceso, con las otras dos variables *acces* y *acces_secret*, mediante el método *set_acces_token.* Por último, se devuelve un objeto *tweepy.API*, el cual se genera llamando a su constructor con el objeto *auth* que se ha generado previamente.

8.6.1.3 TweetAnswerer

TwitterAnswerer tiene las funciones de responder a los *tweet*s que indica la clase *TweetFinder*.

Tiene un único método *answer,* el cual recibe como parámetros la API de autenticación, la lista que incluye el texto a responder y el enlace para más información, y el id de la mención. Con toda esta información se llama al método *update_status* para publicar el *tweet* de respuesta.

8.6.1.4 Main

Main es un archivo con un único método *main* que llama al método *authenticate* de *TwitterAuth* para obtener el objeto API y después en un bucle infinito hasta que se quiera detener la ejecución al método findNext*Tweet* con la API como parámetro.

8.6.2 Preprocesamiento

En el subsistema de preprocesamiento existen las siguientes clases:

8.6.2.1 PreprocessingHandler

Es la clase encargada de declarar los pasos que se van a seguir en el preprocesamiento. Tiene un único método ***handle*** que llama al método *process* del *PreProcess* que quiera realizar y devuelve el resultado de este método.

8.6.2.2 PreProcess, Lemmatizer, LowerCase y Stopwords

En la clase *PreProcess* se declara la interfaz que deben seguir los pasos de preprocesamiento, el método ***process*** será el común a todas las clases que lo hereden. Recibirá como parámetro el texto a procesar.

Lemmatizer crea un lematizador del tipo *WordNetLemmatizer*, *tokeniza* mediante NLTK el texto y después utiliza nuestro lematizador para extraer el lema de cada *token*. Devuelve el texto como una única cadena.

LowerCase llama al método *lower* de Python que convierte una cadena de caracteres en minúscula.

Stopwords utiliza el paquete *stopwords* de NLTK.corpus, *tokeniza* las palabras del texto y comprueba si estas se encuentran dentro de las *stopwords* ofrecidas, de no ser así se añaden a la cadena que se devolverá.

8.6.2.3 Multi

Multi es una clase que implementa *PreProcess*, pero se utiliza para implementar el patrón *composite* [56] en nuestro preprocesamiento.

En su constructor recibe una lista de *PreProcess* como parámetro y en su método *process* la recorre y llama al *process* de cada una.

8.6.3 *Classifier*

El subsistema de clasificación tiene una única clase que es *Classifier*. En ella se encuentran los siguientes métodos:

PredictCategory es el método que se encarga del trabajo con el modelo entrenado. Recibe el texto como parámetro, carga el vectorizador y el modelo, transforma el texto con el vectorizador y realiza la predicción con el modelo. Devuelve tanto la predicción como la probabilidad que el modelo asigna a ella.

Classifiy recibe como parámetro el texto, llama a los métodos *predictCategory* y *printCategory*, crea una lista con la información obtenida en ambos métodos y la devuelve.

PrintCategory es un método que se utilizó para comprobar el correcto funcionamiento del modelo, recibe una categoría (un entero) como parámetro y devuelve el texto equivalente a su temática.

8.6.4 Subsistema de APIs

El subsistema de APIs consiste en un manejador y un patrón *Strategy* que siguen las distintas APIs.

8.6.4.1 ApiHandler

ApiHandler es una clase con un único método llamado *getAPIFromCategory*, que recibe como parámetros la categoría y la seguridad que tiene el modelo en su predicción, este se encarga de decidir que APIs serán utilizadas para que temáticas. Lo decide de la siguiente forma:

- La primera comprobación es si la seguridad del modelo es menor del 30%, de ser así se elige como API Google, ya que, si nuestro modelo no está seguro de que temática es, lo mejor es usar la API más generalista.
- Después se comprueba si pertenece a las categorías de Ciencia, Educación, Negocios o Política y de ser así, se utilizará la API de Wikipedia.
- Por último, se comprueba si pertenece a la temática de Salud, y si es así se utilizará la API de NHS.
- De no haber parado la ejecución en ninguna de las comprobaciones previas, la API seleccionada será la de Google.

8.6.4.2 Interfaz API, GoogleAPI, WikipediaAPI y NHSAPI

La interfaz API es la que declara el contrato que deben seguir las clases que adapten APIs de nuestro sistema. Deberán tener un método *search* que reciba como parámetros el texto a buscar.

GoogleAPI y *NHSAPI* funcionan de forma similar, ya que ambas utilizan el motor de búsqueda de Google. Primero inicializan el servicio, después se llama al método *list* de éste, el cual nos devuelve una lista con todos los resultados, a partir de la cual se elige el primero y de él se extraen el *snippet* y el enlace a la página.

La clase *WikipediaAPI* funciona llamando al método *search* del paquete Wikipedia de Python, lo que nos devuelve un objeto *page*, del cual extraemos el enlace y el *summary* (resumen). Como paso extra, se utiliza un método *deletePhonetics*, que elimina las partes del texto que habitualmente tienen las entradas de Wikipedia en el cual se ponen los fonemas que componen los nombres, ya que no es una información considerada necesaria. Este método funciona a grandes rasgos eliminando los primeros paréntesis que encuentra después del nombre de la página.

Capítulo 9. Desarrollo de las Pruebas

En esta sección se presentará cómo se han desarrollado las pruebas propuestas previamente.

9.1 Pruebas Unitarias

A continuación, se muestran los resultados de las pruebas unitarias realizadas.

9.1.1 Pruebas unitarias de la interfaz con Twitter

Primero se expondrán los resultados del subsistema de Twitter. Hay que destacar que, por su naturaleza, para comprobarlo no es necesario introducir datos previos para hacer las comprobaciones.

Responsable	Prueba	Resultado Obtenido
TwitterAuth	Conexión mediante tweepy a la API de Twitter con las credenciales de nuestra cuenta	El sistema es capaz de conectarse a Twitter y nos devuelve el objeto del tipo *tweepy.api* con el que poder trabajar.
TweetFinder	Recopilación de *tweet*s	Se recopilan todos los *tweet*s en los que se ha mencionado a nuestra cuenta.
TweetFinder	Distinción de *tweet*s respondidos	Sólo se devuelven los *tweet*s posteriores al último *tweet* respondido, guardado en el *last_id*, o en caso de que ese archivo no se encuentre, los *tweet*s posteriores al arrancado del sistema.
TweetFinder	*Tweet* nuevo de un usuario	Con el sistema ya arrancado, se detecta un nuevo *tweet* de un usuario.
TweetAnswerer	Responder a un *tweet*	El sistema es capaz de responder a un *tweet* de un usuario una sola vez.
TweetAnswerer	Guardar último id respondido	Cuando se ha respondido a un usuario se guarda el id del *tweet* al que se responde como *last_id*.

Tabla 51. Resultado Pruebas Unitarias Twitter.

9.1.2 Pruebas unitarias preprocesamiento

En el preprocesamiento, en cambio, sí tenemos unos datos previos que realizar.

Responsable	Prueba	Datos Introducidos	Resultado
Stopwords	Eliminación de palabras vacías	"This is a sample for testing purposes"	"sample testing purposes"
toLowerCase	Conversión a minúscula	"This is a sample for testing purposes"	"this is a sample for testing purposes"
Lemmatizer	Lematización	"This is a sample for testing purposes"	"This is a sample for *testing* purpose"
Multi	Combinaciones de los pasos anteriores	"This is a sample for testing purposes"	"sample testing purpose"

Tabla 52. Resultado Pruebas Unitarias Preprocesamiento.

Si bien la prueba del Lemmatizer si ha lematizado *purposes*, no lo ha hecho con *testing*, donde en el caso ideal habríamos obtenido *test*.

9.1.3 Pruebas unitarias del Clasificador

El *Classifier* será probado en mayor profundidad en el siguiente apartado de pruebas de integración.

Responsable	Prueba	Resultado Obtenido
Classifier	Categorizar una pregunta	El *Classifier* recibe el texto y devuelve la temática.

Tabla 53. Resultado Pruebas Unitarias Clasificador.

Se observa que el funcionamiento de la clase *Classifier* es el esperado, siendo capaz de devolver una categoría ante un texto introducido.

9.1.4 Pruebas unitarias del módulo de APIs

Se realizará una prueba por API para comprobar su funcionamiento individual.

Responsable	Prueba	Datos Introducidos	Respuesta obtenida
WikipediaAPI	Responder con la API de Wikipedia	Barack Obama	'Barack Hussein Obama II is an American retired politician who served as the 44th president of the United States from 2009 to 2017. A member of…', 'https://en.wikipedia.org/wiki/Barack_Obama'
NHSAPI	Responder con la API de NHS	Headache	'If painkillers do not help and your headache gets worse; you have a bad throbbing pain at the front or side of your head – it could be a migraine or, more rarely...', 'https://www.nhs.uk/conditions/headaches/'
GoogleAPI	Responder con la API de Google	Famous songs from the Beatles	The Top Ten Beatles Songs of All Time · "While My Guitar Gently Weeps" · "A Day in the Life" · "I Want to Hold Your Hand" · "Strawberry Fields...', 'https://www.rollingstone.com/culture/culture-lists/the-top-ten-beatles-songs-of-all-time-10664/'

Tabla 54. Pruebas Unitarias APIs.

Como se pude observar, las tres APIs con capaces de generar una respuesta ante cada petición, devolviendo también el enlace a la página original de la cual se extrajo la información.

9.2 Pruebas de Integración, del Sistema y de Rendimiento

A continuación, se muestran los resultados de las pruebas de integración. Posteriormente, se explicará en detalle las tres últimas pruebas en las que se evalúan los tres tipos de funcionamiento distintos de la aplicación.

El resultado de las pruebas funcionales ha sido el siguiente:

Prueba	Datos introducidos	Resultado Obtenido
Preprocesar el contenido de un *tweet*	*What are the best places to eat in Spain?*	best place eat spain
Clasificar un *tweet* como Salud	*How can I stop feeling dizzy?*	Salud
Clasificar un *tweet* como Política	*Who is the president of Uruguay?*	Política
Clasificar un *tweet* como Deportes	*Who is first in the Premier League*	Deportes
Clasificar un *tweet* como Finanzas	*What's the average income in Europe?*	Finanzas
Clasificar un *tweet* como Historia	*When did the French Revolution Start?*	Educación
Clasificar un *tweet* como Ciencia y Matemáticas	*What's a black hole?*	Ciencia y Matemáticas
Clasificar un *tweet* como Familia y Relaciones	*How can i get new friends?*	Familia y Relaciones
Clasificar un *tweet* como Educación	*What's the largest mammal on earth?*	Ciencia y Matemáticas
Clasificar un *tweet* como Informática	*Wich ones are the most used programming languages?*	Informática
Clasificar un *tweet* como Entretenimiento y Música	*How many Harry Potter Books are there?*	Entretenimiento y Música
Clasificar un *tweet* como Sociedad	*Who are the Kardashians?*	Familia y Relaciones
Responder a un *tweet* de Salud	*How can I stop feeling dizzy?*	Respondido
Responder a un *tweet* de Historia, Ciencia o Política	*What's a black hole?*	Respondido
Responder a un *tweet* de otra temática	*How can i get new friends?*	Respondido

Tabla 55. Resultados pruebas de integración.

Como se puede observar el clasificador realiza una tarea correcta a la hora de categorizar las preguntas. Si bien las categorías de Historia, Educación y Sociedad no fueron clasificadas como se esperana, la temática ofrecida como respuesta tampoco se puede considerar incorrecta, además, a nivel funcional, las preguntas seguirían utilizando las mismas APIs que si se hubiese predicho la categoría deseada.

9.2.1 Respuestas obtenidas

Se procederá ahora con los resultados de las preguntas realizadas en la clasificación.

Ante la pregunta de **Salud**, en la cual preguntamos cómo podemos dejar de sentirnos mareados, se responde con una serie de indicaciones a seguir y el enlace a la página de NHS, por lo que el sistema ha identificado correctamente el tipo de pregunta y además ha dado una respuesta coherente.

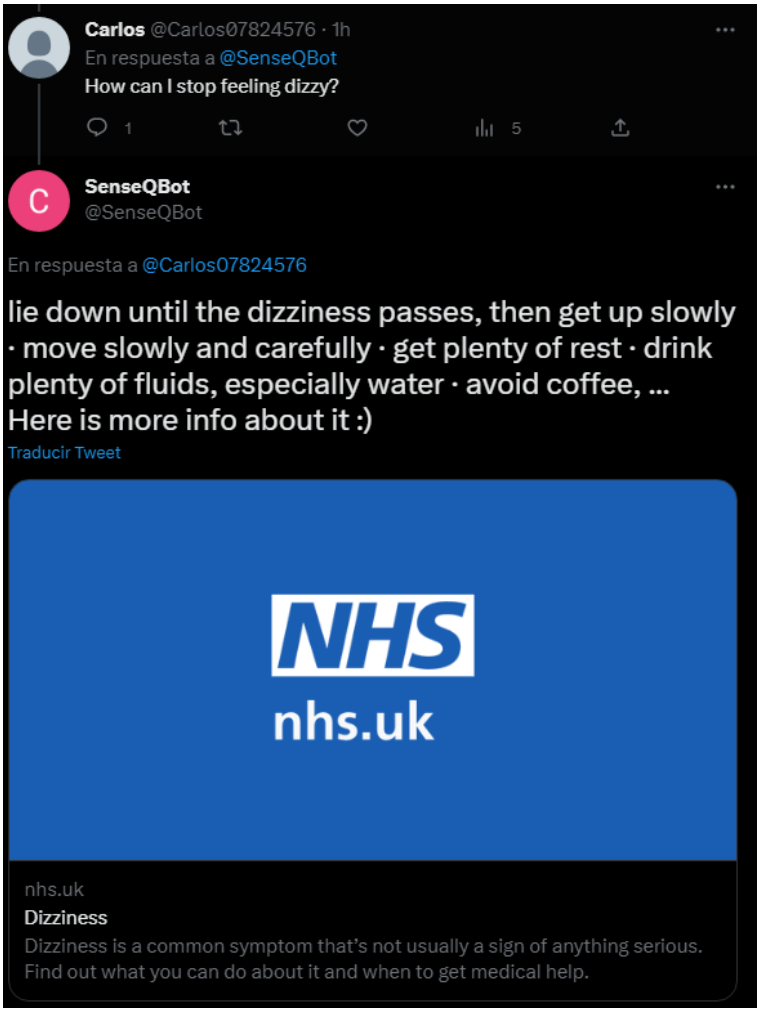

Imagen 56. Respuesta a una pregunta de Salud.

En la pregunta de **Política** preguntábamos por el actual presidente de Uruguay. Es cierto que el sistema detecta correctamente la temática, si bien la respuesta que ofrece no contiene la respuesta a la pregunta. Sin embargo, la respuesta si se encuentra en el enlace que se envía, además de aparecer la foto del presidente.

Imagen 57. Respuesta a pregunta política.

Algo similar pasa con la pregunta de **Finanzas**: no se da la respuesta concisa, pero si se aporta un enlace que nos lleva a los salarios medios de Europa ordenados por países.

Imagen 58. Respuesta a pregunta de Finanzas.

La pregunta de **Historia** había sido clasificada como Educación, lo cual no influye en la forma en la que lo trata nuestro sistema, observamos que se ofrece una respuesta a la pregunta de cuándo empezó la Revolución Francesa.

Imagen 59. Pregunta de historia.

Las preguntas de **Ciencia y Matemáticas** son quizá las que mejor se adaptan al formato de nuestro sistema cuando se utiliza la API de Wikipedia, pues observamos que ante la pregunta de qué es un agujero negro, se responde brevemente con la definición y el enlace por si se quiere seguir leyendo.

Imagen 60. Pregunta de Ciencia.

La última categoría que utiliza la API de Wikipedia es la de **Educación,** en este caso, se especifica como respuesta que la pregunta puede ser respondida de varias formas, y se envía el enlace para que el usuario encuentre la respuesta.

Imagen 61. Pregunta de Educación.

Ahora se verán las preguntas de temáticas más generales, empezando por **Familia y Relaciones**. En ésta se observa como el sistema ofrece una respuesta que se adapta a la petición del usuario.

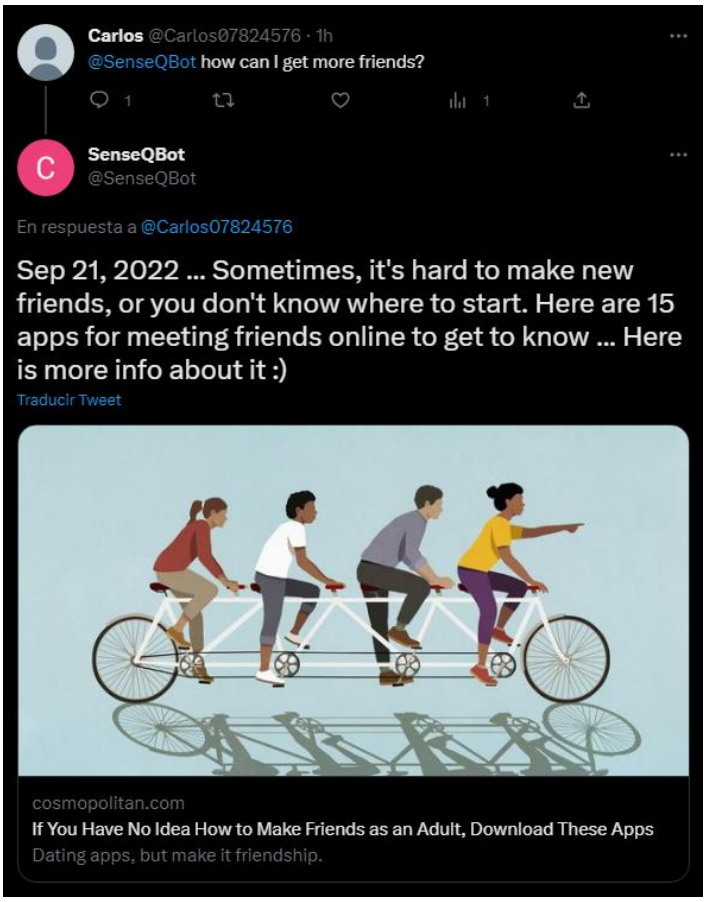

Imagen 62. Pregunta de Familia y Relaciones.

En la pregunta de **Entretenimiento** preguntamos por la cantidad de libros que pertenecen a la saga Harry Potter, de nuevo, la respuesta es acertada.

Imagen 63. Pregunta de Entretenimiento.

La pregunta de **Deportes** es la que con diferencia peor respuesta ha obtenido, si bien ha sido clasificada correctamente, la respuesta que ofrece no es quien es el primero en la Premier League si no cuando fue la primera Premier League.

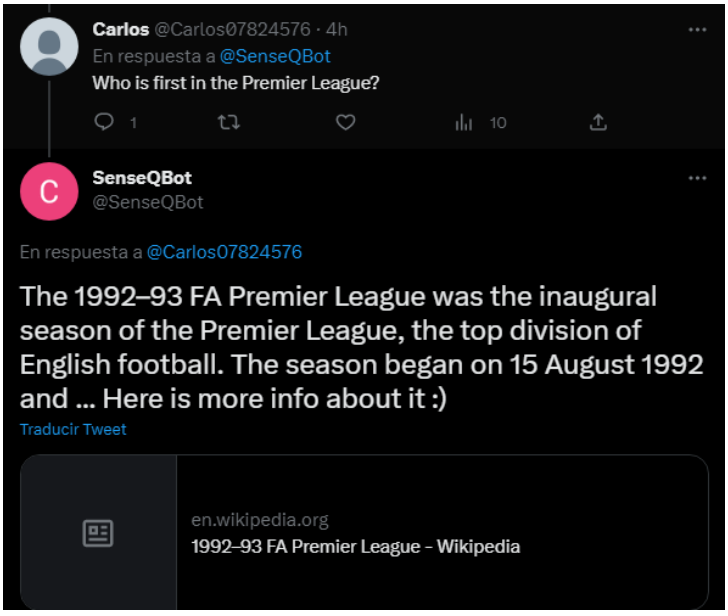

Imagen 64. Pregunta de Deportes.

La pregunta de **Sociedad** fue categorizada erróneamente, pero de nuevo, no de forma en la que altere cómo se pretende que el sistema la trate. El sistema ofrece una respuesta correcta.

Imagen 65. Pregunta de Sociedad.

Por último, ante la pregunta **Informática** de cuáles son los lenguajes de programación más usados se responde con una lista ordenada de ellos.

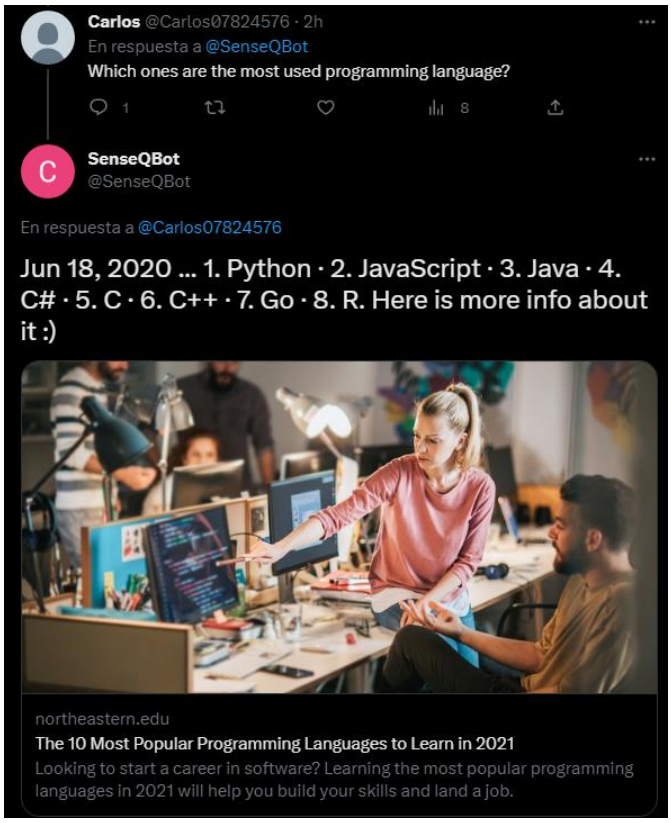

Imagen 66. Pregunta Informática.

Se puede concluir que el sistema es capaz de ofrecer, en la mayoría de los casos, respuestas coherentes ante las preguntas de los usuarios, a pesar de que algunas veces para obtener la respuesta exacta a la cuestión deben acudir al enlace que se les provee.

9.3 Pruebas de rendimiento

En términos de rendimiento el sistema funciona de forma fluida, siendo el mayor impedimento en cuanto al tiempo de respuesta el tiempo de espera que se realiza entre cada comprobación sobre la llegada de nuevos *tweets*. En la siguiente tabla se estudia lo que tarda cada subsistema en ejecutarse. En este caso, se ha puesto el *tweet* previamente al arranque del sistema.

Subsistema	Tiempo
Twitter (Encontrar el *tweet*)	0.34 segundos
Preprocesamiento	0.98 segundos
Clasificación	0.16 segundos
Búsqueda *WikipediaAPI*	1,65 segundos
Búsqueda *GoogleAPI*, NHS API	0.50 segundos
Twitter (Publicar respuesta)	0.37 segundos

Tabla 56. Tiempos de respuesta por subsistema.

Se deduce que los subsistemas más lentos son los de preprocesamiento y búsqueda, con una diferencia considerable entre las búsquedas con la API de Wikipedia y las otras.

En total, el sistema tarda desde que encuentra el *tweet* hasta que lo responde unos 3 segundos de media, esto es, en el mejor de los casos, en el cual el bucle para buscar las menciones se inicia simultáneamente a la llegada del *tweet*. Si no, el usuario puede esperar en el peor de los casos, un máximo de 33 segundos, suponiendo que escriba el *tweet* justo en el momento en el que se inicia el periodo de espera de 30 segundos antes de realizar una nueva pasada.

Capítulo 10. Manuales del Sistema

En este capítulo se detallarán los manuales del sistema relativos a la instalación, el usuario y el programador.

10.1 Instalación y Ejecución

Para ejecutar el programa en nuestra máquina necesitaremos haber instalado Python previamente, en este caso la versión utilizada es la 3.9.0.

Si ya lo tenemos debemos instalar los paquetes que utiliza el proyecto, se ha creado el archivo *requirements.txt* con los requisitos para la ejecución de nuestro programa.

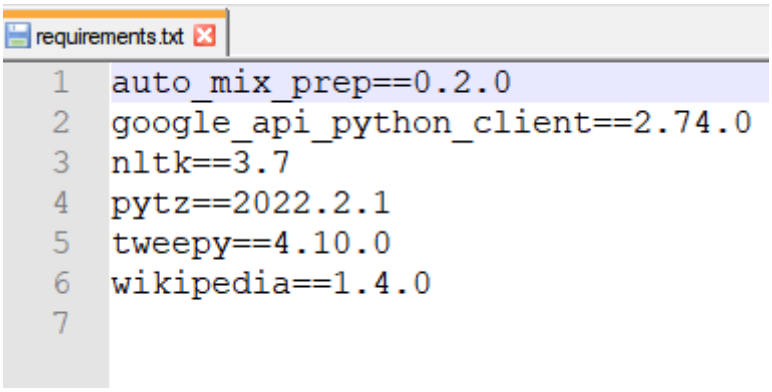

Imagen 67. Archivo Requirements.txt.

Para ejecutar el programa simplemente debemos ejecutar desde la consola el comando:

Python ./*Main*.py

Con esto el programa empezará a responder a los *tweet*s que aún no ha tratado y permanecerá pendiente de la llegada de nuevos para ser respondidos.

10.2 Manual del programador

En el caso del programador, podemos explicar ciertos aspectos de cara a la modificación o expansión del sistema. Principalmente en los subsistemas de preprocesamiento, clasificación y APIs.

10.2.1 Subsistema de preprocesamiento

El subsistema de preprocesamiento sigue dos patrones de diseño, por un lado, el patrón *Strategy* [55] permite añadir nuevos pasos tan sólo creando una clase nueva con el

procedimiento a seguir, esta clase debe heredar de *PreProcess* y tener implementado el método *process*.

Si después queremos añadirlas a la ejecución, lo debemos incluir en el método *handle* de *PreProcessingHandler*, en el cual se encuentran los pasos que queremos seguir.

En la clase *Multi* se ha implementado un patrón *composite* [56], con la idea de poder modificar el número o el orden de preprocesamiento que queremos realizar sin que el sistema se tenga que preocupar de tratarlo de forma diferente.

10.2.2 Subsistema de APIs

Un caso parecido ocurre en el subsistema de APIs, donde encontramos un patrón *Strategy*, por el cual para añadir nuevas APIs debemos heredar de la clase *Api* e implementar el método *search*. Después, en la clase *ApiHandler* debemos modificar el método *getAPIFromCategory* para incluir en qué casos queremos que se utilice la nueva API que se ha habilitado.

10.2.3 Clasificador

Si queremos modificar el clasificador por otro que hayamos generado debemos realizar los siguientes pasos:

1. Tener un clasificador entrenado para categorizar texto según la temática.
2. Guardar, preferiblemente mediante *pickle* ya que es el implementado en el sistema, tanto el clasificador como el vectorizador utilizado para el entrenamiento.
3. En la clase *Classifier* en el método *predictCategory* modificar los atributos *vectorizer* y *model* para que carguen nuestro nuevo clasificador en lugar del anterior.
4. Si las categorías que devuelve nuestro clasificador han cambiado sobre las originales debemos modificar el método *getAPIFromCategory*.

10.3 Manual del usuario

Para poder utilizar el sistema el usuario deberá realizar los siguientes pasos:

En su ordenador o dispositivo móvil debe ir o bien a *twitter.com* o bien a la aplicación de Twitter e iniciar sesión o registrarse si no tiene un usuario.

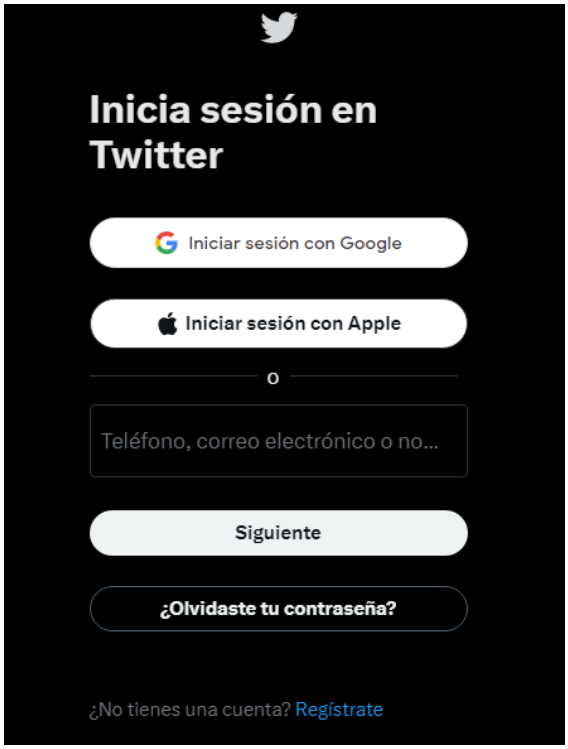

Imagen 68. Manual del usuario. Paso 1: Inicio de sesión.

Una vez introducidas sus credenciales deberá seleccionar la opción de escribir un nuevo *tweet*.

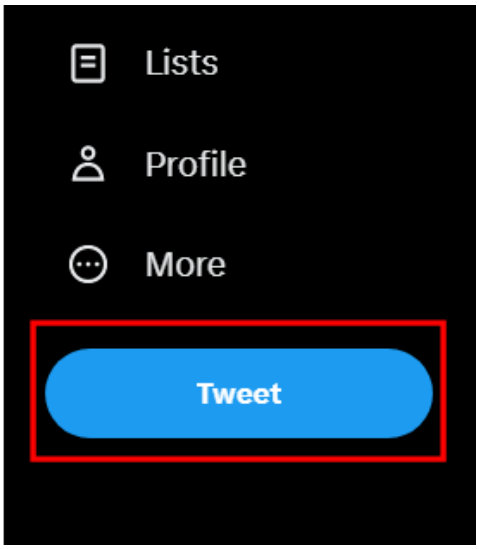

Imagen 69. Manual del usuario. Paso 2: Escribir un nuevo tweet.

Ahora deberá mencionar a la cuenta de SenseQBot, con el @ previo al nombre de usuario, e incluir en el *tweet* la pregunta que quiere realizar y pulsar el botón de *Tweet*/Tuitear.

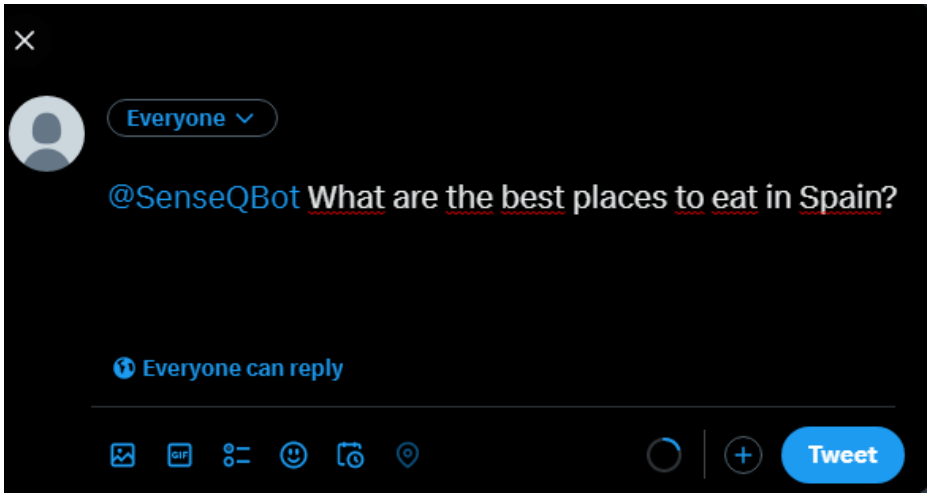

Imagen 70. Manual del usuario. Paso 3: Publicar un tweet.

Ahora debe esperar a que el sistema detecte que le ha mencionado y envíe su respuesta, como máximo, esto puede demorarse 30 segundos.

Llegado ese punto el usuario recibirá una notificación por parte de Twitter de que ha recibido una nueva mención. Debe pulsar en el apartado de notificaciones.

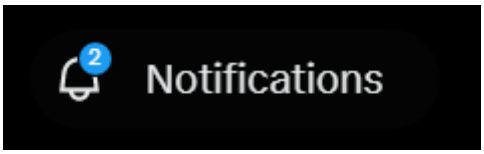

Imagen 71. Manual del usuario. Paso 3: Publicar un tweet.

En esta pestaña ya podrá ver la respuesta que le ha enviado el sistema, y si desea más información, pulsar en el enlace que le lleva a más información.

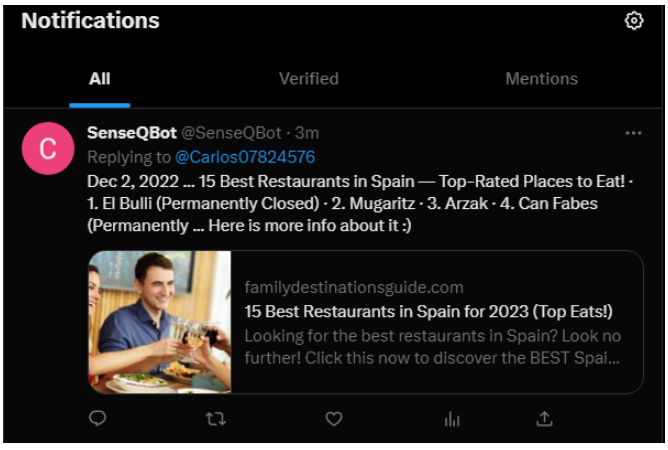

Imagen 72. Manual del usuario. Paso 3: Publicar un tweet.

Capítulo 11. Conclusiones y Ampliaciones

En este apartado se discutirán las conclusiones que se han sacado de la realización de este proyecto, así como posibles ampliaciones futuras.

11.1 Conclusiones

En este proyecto se ha conseguido generar un modelo basado en *machine learning* capaz de clasificar texto según su categoría, con esto, se ha realizado un sistema mediante el cual, un usuario de la red social Twitter puede enviar una pregunta y el sistema es capaz de categorizarla, para así, mediante el uso de diferentes APIs que mejor se ajusten a la temática, responderla de forma coherente.

Durante la realización del proyecto se ha aprendido a crear una cuenta automatizada en Twitter capaz de extraer el texto de las menciones que recibe y ejecutar sobre el procesamiento que queramos.

De la misma forma, se han aprendido nuevos conceptos de *machine learning* y procesamiento de lenguajes naturales. Por ello, se ha destinado una gran parte del tiempo a la investigación sobre este tema.

Igualmente, se aprendieron formas diversas de generar modelos en función del algoritmo que define la clasificación, y se descubrió de qué forma se podían optimizar los resultados para nuestro caso.

En comparación con las alternativas estudiadas, se puede concluir que se ha creado un sistema con un funcionamiento que previamente no se encontraba en la red social. Fuera de ella, en comparación con otros chatbots basados en inteligencia artificial, lo que diferencia a este proyecto sobre ellos es la intención y la forma en la que se realiza la comunicación con el usuario. En proyectos como ChatGPT se busca mantener una conversación fluida, además se utiliza *machine learning* no sólo para entender la pregunta si no para realizar la respuesta, lo que puede hacer que se altere la veracidad de esta. En el caso que nos atañe, en cambio, la interacción con el usuario es mucho más directa e impersonal, y la respuesta no se genera mediante un aprendizaje del sistema, si no que se ofrece siempre una escrita previamente por una persona rea, es decir, se dan respuestas “oficiales” en lugar de deducciones realizadas por la máquina..

A nivel personal, este proyecto ha sido una gran experiencia que ha hecho que aprenda una gran cantidad de conocimientos en campos que resultaban prácticamente desconocidos para mí. Por un lado, Python era un lenguaje que no había utilizado apenas previamente, y realizar un proyecto de esta envergadura me ha servido para descubrir las ventajas y desventajas que supone como lenguaje de programación. Por otro, todo el proceso de investigación, búsqueda

de artículos y documentación, tanto de *machine learning* como de las APIs de Twitter, Wikipedia, Google… Ha supuesto un reto diferente al que estaba acostumbrado y lo considero una buena forma de trabajar que si bien, requiere de más tiempo, es mucho más eficiente de cara a conseguir resultados a la par que muy enriquecedora de cara al futuro.

11.2 Ampliaciones

A continuación, se mencionan algunas de las posibles ampliaciones que se pueden realizar en nuestro proyecto.

11.2.1 Nuevas APIs

Añadir nuevas APIs como, por ejemplo, una API para las preguntas deportivas haría que nuestro sistema mejorase en las respuestas ofrecidas en esa temática, para ello se debe buscar una API que cumpla los requisitos que queremos.

Para ello, se debe añadir la nueva API como se ha explicado en el apartado *Manual del programador* y modificar la clase *ApiHandler* para que incluya nuestra nueva casuística.

Un aspecto para tener en cuenta, y uno de los motivos por el que no se ha incluido una como la del ejemplo, es que las preguntas de una temática pueden variar mucho en lo que piden, en nuestro caso de ejemplo, una pregunta deportiva puede ser tanto un resultado de un encuentro, para lo cual necesitaríamos una API que ofrezca resultados deportivos, como una pregunta sobre una noticia reciente, para lo que necesitaríamos una que nos ofrezca noticias.

11.2.2 Modelos entrenados con otros *datasets* o algoritmos

Otra posible ampliación que se puede realizar es entrenar un nuevo modelo con un *dataset* diferente para luego tratar de acoplarlo al antiguo, con la finalidad de mejorar el acierto que tiene nuestro sistema a la hora de realizar las clasificaciones.

Hay que tener en cuenta que un *dataset* distinto es más probable que esté formado por distintas categorías a las de la colección anterior, por lo que debemos adaptarlo. Si bien esto podría ser sencillo, ya que tan solo habría que buscar las equivalencias entre las categorías nuevas y las viejas, por ejemplo, si el *dataset* nuevo tiene una categoría *Medicine* (Medicina) tan solo debemos asegurarnos de que el valor que se devuelve cuando el modelo la detecta es el mismo que el devuelto en nuestro modelo original cuando se detecta la clase *Health* (Salud).

Otra ampliación posible sería, en caso de encontrar un algoritmo diferente a los ya probados que aporte un mejor nivel de acierto, añadirlo al modelo realizado mediante *Stacking*, con la idea de aumentar su eficiencia.

11.2.3 Preprocesamiento

El preprocesamiento como se explicó en el apartado *Manual del programador* es fácilmente ampliable, por lo que el desarrollador podría añadir nuevos pasos a este apartado para buscar un mejor rendimiento del sistema.

Capítulo 12. Planificación del Proyecto y Presupuesto finales

A continuación, se verá la planificación y presupuesto finales del proyecto, y se comparará con sus versiones iniciales.

12.1 Planificación Final

En este apartado se detallará la planificación final del proyecto, observando las diferencias con respecto a la inicial.

Hay varios cambios que se realizaron con respecto a la planificación inicial, el primero es el desglose más detallado a la hora de las tareas, conforme avanzamos en la realización del proyecto, nos damos cuenta de que la planificación original se queda muy poco descriptiva con respecto al trabajo que se debe realizar. Además, aparecen nuevas tareas que no estaban incluidas originalmente como la generación y comparativa de resultados. Otro cambio que destacar es la medida del tiempo, se pasó de días de trabajo a horas, ya que no todos los días era posible trabajar la misma cantidad. En cuanto a la diferencia de esta, el tiempo inicial era de 49 días laborales, es decir 392 horas, la duración final fue de 373 horas, una diferencia de un 5%. Si bien las horas finales no varían tanto de lo planificado inicialmente, la división de esas horas en los distintos trabajos sí que se ve claramente afectada, lo cual, puede generar problemas durante el desarrollo si el equipo está compuesto por más de una persona.

Con respecto a los recursos se modifican de varios a tan sólo uno, al cual le asignamos una tarifa estándar por horas.

Se desarrolla ahora la planificación siguiendo cada módulo individualmente para mayor claridad.

12.1.1 Investigación Previa

La planificación original para este apartado era de 136 horas (17 días), se ha incrementado en gran medida, ya que aparece el apartado de comparativa de modelos, con el que inicialmente no se contó y que además llevó un tiempo considerable.

Tarea	Duración	Recurso
Investigación Previa	**254 horas**	
Búsqueda proyectos similares	20 horas	Carlos
Búsqueda de artículos y referencias	30 horas	Carlos
Búsqueda de *datasets*	20 horas	Carlos
Búsqueda de APIS	25 horas	Carlos
Herramientas	**83 horas**	
Python	15 horas	Carlos
Twitter API	10 horas	Carlos
Tweepy	8 horas	Carlos
NLTK	10 horas	Carlos
Scikit-Learn	10 horas	Carlos
WikiMedia API	15 horas	Carlos
GoogleProgrammable*Search*	15 horas	Carlos
Comparativa de modelos	**76 horas**	
Selección de *Datasets*	8 horas	Carlos
Comparativas Preprocesamiento	20 horas	Carlos
Generación de modelos	40 horas	Carlos
Comparativa de modelos	8 horas	Carlos

Tabla 57. Planificación final Investigación previa.

12.1.2 Análisis

En el apartado de análisis se cambia “Documentación” por el análisis de los casos de uso, para este módulo se realizó una predicción demasiado generosa inicialmente, ya que de las 80 horas (10 días) supuestas, sólo se necesitaron 10. Esto se debe principalmente a que gran parte del trabajo que se pensó para esta tarea, se terminaron realizando previa o posteriormente.

Tarea	Duración	Recurso
Análisis	**10 horas**	
Requisitos	5 horas	Carlos
Casos de Uso	5 horas	Carlos

Tabla 58. Planificación Análisis.

12.1.3 Módulos

Para los módulos en origen se estimaron 56 horas (9 días) con cada módulo necesitando 24 horas (3 días) para su realización, el resultado final no se alejó en exceso de la planificación, con una duración real de 65 horas.

Tarea	Duración	Recurso
Módulos	**65 horas**	
Twitter	**22 horas**	
Diseño	2 horas	Carlos
Twitter *Auth*	3 horas	Carlos
TweetFinder	10 horas	Carlos
TweetAnswerer	5 horas	Carlos
Pruebas	2 horas	Carlos
Preprocesamiento	**20 horas**	
Diseño	1 hora	Carlos
Interfaz *PreProcess* y subestrategias	10 horas	Carlos
PreProcessing Handler	3 horas	Carlos
Integración	2 horas	Carlos
Pruebas	4 horas	Carlos
Clasificación	**8 horas**	
Diseño	1 hora	Carlos
Classifier	3 horas	Carlos
Integración	1 hora	Carlos
Pruebas	3 horas	Carlos
Interfaz de APIs	**15 horas**	
Diseño	1 hora	Carlos
ApiHandler	1 hora	Carlos
WikipediaAPI	5 horas	Carlos
Google y NHS API	4 horas	Carlos
Integración	1 hora	Carlos
Pruebas	3 horas	Carlos

Tabla 59. Planificación final Módulos.

12.1.4 Pruebas Finales y Documentación

Las pruebas finales cambian en su contenido con respecto a la planificación inicial, dividiéndose en los dos tipos de pruebas realizados, el tiempo estimado era de 27 horas (3 días), un valor no muy alejado de las 24 horas que se terminaron necesitando.

Tarea	Duración	Recurso
Pruebas Finales	**24 horas**	
Pruebas Unitarias	10 horas	Carlos
Pruebas de Integración	14 horas	Carlos
Documentación	**60 horas**	Carlos

Tabla 60. Planificación Peuebas y documentación.

Con respecto a la documentación, inicialmente estaba dentro del análisis, pero dado que eso tan sólo conformaba una parte de ella, se ha sacado como su propia tarea. Dentro de esa planificación inicial se le asignaban 80 horas (10 días), algo superior a las 60 que finalmente se necesitaron.

12.2 Presupuesto Final

En esta sección se expondrá el presupuesto final de nuestro proyecto, primero veremos la versión simplificada para compararla con la inicial, después realizaremos un desglose por tareas.

Con respecto a los materiales, no se han necesitado más de los inicialmente presupuestados, por lo que ese apartado se mantiene igual.

Material	Precio
Ordenador	500 €
Microsoft Office	60 €
Total	**560 €**

Tabla 61. Presupuesto de materiales final.

A pesar de haber unificado los recursos en uno solo, vamos a mantener las tasas por hora de cada uno en el cálculo de nuestro presupuesto. El coste original para el proyecto con los recursos iniciales era de 6.064 € sin el prorrateo del beneficio ni el IVA.

El resumen de nuestro presupuesto es el siguiente:

Nombre	Coste
Investigación Previa	2.889,00 €
Análisis	120,00 €
Módulos	1.015,00 €

Twitter	362,00 €
Preprocesamiento	313,00 €
Clasificación	107,00 €
Interfaz de APIs	425,00 €
Pruebas Finales	192,00 €
Documentación	660,00 €
Materiales	560,00 €
TOTAL	5.436,00 €

Tabla 62. Presupuesto final resumido.

En el caso del presupuesto, se ve reducido 600 € con respecto al presupuesto original, pero como ocurre con la planificación el reparto de éste dentro de los distintos apartados varía en gran medida con respecto a lo inicialmente planteado. Si se hubiese realizado el proyecto con el presupuesto anterior, la fase de investigación por ejemplo habría agotado su presupuesto mucho antes de su finalización.

Se evaluará ahora en detalle cada apartado de la planificación.

12.2.1 Presupuesto Investigación Previa

Como dijimos anteriormente, el presupuesto para la Investigación previa es el que se ve más modificado con respecto al inicial, con una diferencia de 1.410 €

Nombre	Coste
Investigación Previa	2.899,00 €
Búsqueda proyectos similares	240,00 €
Búsqueda de artículos y referencias	360,00 €
Búsqueda de datasets	240,00 €
Búsqueda de APIS	300,00 €
Herramientas	913,00 €
Python	165,00 €
Twitter API	110,00 €
Tweepy	88,00 €
NLTK	110,00 €
Scikit-Learn	110,00 €
WikiMedia API	165,00 €
GoogleProgrammable*Search*	165,00 €
Comparativa de modelos	836,00 €
Selección de Datasets	88,00 €
Comparativas Preprocesamiento	220,00 €

Generación de modelos	440,00 €
Comparativa de modelos	88,00 €

Tabla 63. Presupuesto final investigación previa.

12.2.2 Presupuesto Análisis

El presupuesto final para el análisis se ve drásticamente reducido ya que gran parte de las tareas que se consideraban en ellas ahora se encuentran en otros apartados.

Nombre	Coste
Análisis	120,00 €
Requisitos	60,00 €
Casos de Uso	60,00 €

Tabla 64. Presupuesto final.

12.2.3 Presupuesto Módulos

EL presupuesto para los subsistemas también se ve reducido ya que, en este punto con la investigación inicial realizada, el desarrollo de estos módulos fue más rápido de lo esperado.

Nombre	Coste
Módulos	675,00 €
Cuenta de Twitter	362,00 €
Diseño	22,00 €
Twitter *Auth*	54,00 €
TweetFinder	180,00 €
TweetAnswerer	90,00 €
Pruebas	16,00 €
Preprocesamiento	313,00 €
Diseño	11,00 €
Interfaz *PreProcess* y subestrategias	180,00 €
PreProcessing Handler	54,00 €
Integración	36,00 €
Pruebas	32,00 €
Clasificación	107,00 €
Diseño	11,00 €
Classifier	54,00 €
Integración	18,00 €
Pruebas	24,00 €
Interfaz de APIs	425,00 €
Diseño	11,00 €
ApiHandler	18,00 €
WikipediaAPI	90,00 €

Google y NHS API	72,00 €
Integración	18,00 €
Pruebas	24,00 €

Tabla 65. Presupuesto final módulos.

12.2.4 Presupuesto Pruebas finales y Documentación

Por último, las pruebas finales también ven reducido su presupuesto, y aparece la documentación como tarea propia, previamente dentro del análisis, la cual se aproxima al presupuesto inicial.

Con respecto a las pruebas, al haberse realizado también durante el desarrollo de los módulos, necesitaron un tiempo más reducido del esperado, ya que se tenía ya un conocimiento sólido sobre cómo realizarlas.

Nombre	Coste
Pruebas Finales	192,00 €
Pruebas Unitarias	80,00 €
Pruebas de Integración	112,00 €
Documentación	660,00 €

Tabla 66. Presupuesto final documentación.

Capítulo 13. Referencias Bibliográficas

En este capítulo se listan todas las referencias consultadas a lo largo de este proyecto:

13.1 Libros y Artículos

[1] D. M. Perez Garcia, S. Saffon Lopez, and H. Donis, "Everybody is talking about Virtual Assistants, but how are people really using them?," 2018. doi: 10.14236/ewic/hci2018.96.

[2] Vixen Labs, "Voice Consumer Index 2022." Accessed: Jan. 12, 2023. [Online]. Available: https://vixenlabs.co/research/voice-consumer-index-2022

[14] J. Vasilakes, S. Zhou, and R. Zhang, "Natural language processing," in *Machine Learning in Cardiovascular Medicine*, 2020. doi: 10.1016/B978-0-12-820273-9.00006-3.

[15] ANTONIO MORENO, "Procesamiento del lenguaje natural ¿qué es?," *https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/*.

[16] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, "Explainable Machine Learning for Scientific Insights and Discoveries," *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2020.2976199.

[18] C. S. Wickramasinghe, K. Amarasinghe, D. L. Marino, C. Rieger, and M. Manic, "Explainable Unsupervised Machine Learning for Cyber-Physical Systems," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3112397.

[19] G. Daniel and J. Cabot, "The Software Challenges of Building Smart Chatbots," in *Proceedings - International Conference on Software Engineering*, May 2021, pp. 324–325. doi: 10.1109/ICSE-Companion52605.2021.00138.

[21] J. Lee, "Notes on Naive Bayes Classifiers for Spam Filtering."

[31] X. Zhang and Y. LeCun, "Text Understanding from Scratch," Feb. 2015, [Online]. Available: http://arxiv.org/abs/1502.01710

[33] W. Etaiwi and G. Naymat, "The Impact of applying Different Preprocessing Steps on Review Spam Detection," in *Procedia Computer Science*, 2017, vol. 113, pp. 273–279. doi: 10.1016/j.procs.2017.08.368.

[35] A. Kadhim, "An Evaluation of Preprocessing Techniques for Text Classification Pattern Recognition View project Improvement text classification using log(TF-IDF) with K-NN Algorithm View project." [Online]. Available: https://sites.google.com/site/ijcsis/

[54] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, “Text Classification Using Machine Learning Techniques.”

[61] Naukri, “Why Python For Machine Learning Is Important,” *https://www.naukri.com/learning/articles/why-python-for-machine-learning-is-important/*.

13.2 Referencias en Internet

[3] “Answerbot AI,” *https://twitter.com/answerbot_ai*.

[4] “QnAMaker,” *https://www.qnamaker.ai*.

[5] “Twidge,” *https://atareao.es/software/social/4441/*.

[6] “Chat GPT,” *https://openai.com/blog/chatgpt/*.

[7] T. G. Dietterich, “Machine-learning research: Four current directions,” *AI Mag*, vol. 18, no. 4, 1997.

[8] “What is Java Used for?,” *https://www.coursera.org/articles/what-is-java-used-for*.

[9] “PYPL index,” *https://pypl.github.io/PYPL.html*.

[10] “Tiobe 2023 Index,” *https://www.tiobe.com/tiobe-index/*.

[11] “The best AI Programming Languages, Java vs Python,” *https://huddle.eurostarsoftwaretesting.com/the-best-ai-programming-languages-java-vs-python/#:~:text=Python%20is%20more%20suitable%20for,are%20bugs%20in%20your%20code.*

[12] “Acerca de Ruby,” *https://www.ruby-lang.org/es/about/*.

[13] “Ruby vs Python,” *https://www.coursera.org/articles/ruby-vs-python*.

[17] “¿Qué es Machine Learning?,” *https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico*.

[20] “Naive Bayes Classifiers,” *https://www.geeksforgeeks.org/naive-bayes-classifiers/*.

[22] “Decision Tree Algorithm, Explained,” *https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html*.

[23] “Understanding Random Forest,” *https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/*.

[24] “K-Nearest Neighbor(KNN) Algorithm for Machine Learning,” *https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning*.

[25] "SUPPORT VECTOR MACHINE (SVM) - MÁQUINA DE VECTORES DE SOPORTE," *https://rstudio-pubs-static.s3.amazonaws.com/570352_e34015b16f1a47e883e04c6195d4711f.html*.

[26] "Precision Recall FScore Support," *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html*.

[27] "F1 Score," *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html*.

[28] "WordBank," *http://data.worldbank.org/*.

[29] "Data World," *https://data.world/datasets/open-data*.

[30] "Kaggle," *https://www.kaggle.com*.

[32] ScikitLearn, "Preprocessing," *https://scikit-learn.org/stable/modules/preprocessing.html).* .

[34] "Importance of Text Preprocessing," *https://www.pluralsight.com/guides/importance-of-text-pre-processing* .

[36] "NLTK Stopwords," *https://pythonspot.com/nltk-stop-words/*.

[37] "Text Preprocessing: stopwords removal," *https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a*.

[38] "Main differences between stemming and lemmatization," *https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/*.

[39] "Naive Bayes," *https://scikit-learn.org/stable/modules/naive_bayes.html*.

[40] "MultinomialNB," *https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB*.

[41] "Laplace smoothing in Naïve Bayes algorithm," *https://towardsdatascience.com/laplace-smoothing-in-naïve-bayes-algorithm-9c237a8bdece*.

[42] "Scikit Learn," *https://scikit-learn.org*.

[43] "RandomForestClassifier," *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*.

[44] "Gini index," *https://blog.quantinsti.com/gini-index/*.

[45] "Gini vs Entropy," *https://quantdare.com/decision-trees-gini-vs-entropy/*.

[46] "What is Log Loss," *https://www.kaggle.com/code/dansbecker/what-is-log-loss*.

[47] Scikit Learn, "Tree Modules," *https://scikit-learn.org/stable/modules/tree.html#tree*.

[48] "LinearSVC," *https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html*.

[49] "L2 and hinge combination," *https://github.com/scikit-learn/scikit-learn/issues/7714*.

[50] "KNeighbors Classifier," *https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier*.

[51] "Stacking Classifier," *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html*.

[52] "Logistic Regression," *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression*.

[53] "Predict Proba," *https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB.predict_proba*.

[55] "Patrón Strategy (GoF)," *https://sacavix.com/2020/02/patron-strategy-gof/*.

[56] "Patrón Composite," *https://refactoring.guru/es/design-patterns/composite*.

[57] "Patrón Adapter," *https://refactoring.guru/es/design-patterns/adapter*.

[58] "PEP08," *https://peps.python.org/pep-0008/*.

[59] "Python," *https://www.python.org*.

[60] Wikipedia, "Python," *https://es.wikipedia.org/wiki/Python*.

[61] Naukri, "Why Python For Machine Learning Is Important," *https://www.naukri.com/learning/articles/why-python-for-machine-learning-is-important/*.

[62] "PyScripter," *https://pyscripter.dev*.

[63] Wikipedia, "PyScripter," *https://en.wikipedia.org/wiki/PyScripter*.

[64] "Pycharm," *https://www.jetbrains.com/es-es/pycharm/*.

[65] Universidad Carlos III, "PyCharm," *https://www.uc3m.es/sdic/software/pycharm*.

[66] "Microsoft Project," *https://www.microsoft.com/es-es/microsoft-365/project/project-management-software*.

[67] Wikipedia, "Microsoft Project," *https://es.wikipedia.org/wiki/Microsoft_Project*.

[68] "OperaGX," *https://www.opera.com/es/gx*.

[69] Wikipedia, "Opera," *https://es.wikipedia.org/wiki/Opera_(navegador)#Versiones_para_otros_dispositivos*.

[70] "Twitter," *www.twitter.com*.

[71] Wikipedia, "Twitter," *https://es.wikipedia.org/wiki/Twitter*.

[72] "Putty," *https://www.putty.org*.

[73] Wikipedia, "PuTTY," *https://en.wikipedia.org/wiki/PuTTY*.

[74] "WinSCP," *https://winscp.net/eng/docs/lang:es*.

[75] "WinSCP," *https://es.wikipedia.org/wiki/WinSCP*.

[76] "MobaXTerm," *https://mobaxterm.mobatek.net*.

[77] "MobaXTerm," *https://www.bu.edu/tech/services/security/server/vulnerability-management/xprobe/securing-mobaxterm/#:~:text=MobaXterm%20is%20an%20application%20that,on%20the%20Microsoft%20Windows%20desktop*.

[78] Wikipedia, "Scikit-Learn," *https://es.wikipedia.org/wiki/Scikit-learn*.

[79] "NLTK," *https://www.nltk.org*.

[80] Wikipedia, "NLTK," *https://es.wikipedia.org/wiki/NLTK*.

[81] "Tweepy," *https://www.tweepy.org*.

[82] Morioh, "Tweepy," *https://morioh.com/p/320fc711355d*.

[83] Python Docs, "Pickle," *https://docs.python.org/3/library/pickle.html*.

[84] "Media Wiki," *https://www.mediawiki.org/wiki/MediaWiki/es*.

[85] MediaWiki, "MediaWiki API," *https://www.mediawiki.org/wiki/API:Main_page/es*.

[86] " Google Programmable Search Engine," *https://programmablesearchengine.google.com/about/*.

[87] Wikipedia, "Google Programmable Search Engine," *https://en.wikipedia.org/wiki/Google_Programmable_Search_Engine*.

[88] "Pandas," *https://pandas.pydata.org*.

[89] Wikipedia, "Panda," *https://en.wikipedia.org/wiki/Pandas_(software)*.

[90] "Understanding Vectorizers," *https://sahanidharmendra19.medium.com/understanding-countvectorizer-tfidftransformer-tfidfvectorizer-with-calculation-7d509efd470f*.

[91] "Common Health Questions API NHS," *https://developer.api.nhs.uk/nhs-api/documentation/common-health-questions*.

Capítulo 14. Apéndices

En este capítulo se podrá encontrar memoria del trabajo, entre los que se encuentran, el glosario de términos. un resumen del contenido entregado en el archivo adjunto así como un índice con las actas de las reuniones.

14.1 Glosario de términos

- **Accuracy:** en *machine learning*, exactitud de un modelo, la relación entre el total de aciertos y el total de predicciones.
- **API:** conjunto de procedimientos que ofrece una biblioteca para ser utilizada por otro software.
- **Chatbot:** programa capaz de interactuar con el usuario mediante el lenguaje.
- **Clasificación:** campo dentro de *machine learning* en el cual un modelo recibe una un conjunto de datos nuevos e intenta predecir un valor concreto.
- **Clasificador:** algoritmo capaz de realizar una clasificación.
- **Decision Tree:** algoritmo de creación de un modelo basado en *machine learning* mediante la generación de un árbol de decisiones.
- **F1-Score:** métrica para la evaluación de un algoritmo, media armónica entre *precision* y *recall*.
- **KNeighbors: a**lgoritmo clasificador que realiza predicciones basándose en la proximidad de los datos introducidos a los utilizados para su entrenamiento.
- **Lematización:** obtención de la raíz o lema de una palabra mediante el uso de diccionarios morfológicos.
- **LinearSVC:** algoritmo de clasificación que utiliza máquinas de vectores soporte (SVM).
- **Machine Learning:** campo de la inteligencia artificial que reúne los métodos y técnicas que imitan el aprendizaje.
- **Modelo:** ver Clasificador.
- **NLP:** procesamiento de lenguajes natural.
- **NLTK:** conjunto de librerías y programas para el procesamiento de lenguajes natural.
- **Precision:** métrica de *machine learning* que muestra la relación entre los positivos acertados y el total de positivos.
- **Preprocesamiento:** estandarización del texto previo a su tratamiento con un modelo de *machine learning*.
- **Random Forest:** algoritmo clasificador basado en la creación de varios *Decision Trees*.
- **Recall:** en un clasificador, resultado de dividir los positivos acertados entre los positivos y la suma de éstos con los falsos positivos.
- **Scikit-Learn:** biblioteca de aprendizaje automático de software libre para el lenguaje de programación Python.
- **Stacking:** apilar varios modelos para que actúen como uno solo.
- **Stemming:** obtención de la raíz de una palabra mediante la eliminación automática de sufijos y prefijos.

- **Stopwords:** palabras que no aportan significado a una oración y son filtradas durante un proceso NLP.
- **Support Vector Machine:** conjunto de algoritmos de aprendizaje supervisado dentro del *machine learning*.
- **Tweepy:** biblioteca que permite el uso de Twitter mediante Python.
- **Tweet:** cualquier publicación realizada en Twitter.
- **Twitter:** red social de microblogueo.

14.2 Contenido Entregado en el Archivo adjunto

Adjunto a la documentación se encuentra un archivo con el siguiente contenido:

Directorio	Contenido
./machineLearning	Contiene el código utilizado para la generación de los modelos de *machine learning*, organizados en subcarpetas.
./machineLearning/NaiveBayes	Contiene el código utilizado para la generación de los modelos basados en *Naive Bayes*
./machineLearning/DecisionTree	Contiene el código utilizado para la generación de los modelos basados en *DecisionTree*
./machineLearning/LinearSVC	Contiene el código utilizado para la generación de los modelos basados en *LinearSVC*
./machineLearning/RandomForest	Contiene el código utilizado para la generación de los modelos basados en *RandomForest*
./machineLearning/KNeighbors	Contiene el código utilizado para la generación de los modelos basados en *Kneighbors*
./machineLearning/Stacking	Contiene el código utilizado para la generación de los modelos basados en *Stacking*
./src	Contiene el código fuente y los archivos necesarios para la ejecución del sistema
./src/API	Contiene el código relativo al subsistema de APIs
./src/Classifier	Contiene el código relativo al subsistema de Clasificación
./src/preprocessing	Contiene el código relativo al subsistema de Preprocesamiento
./src/twitterUI	Contiene el código relativo al subsistema de Twitter, se encuentra

	aquí también el archivo *main* que se debe ejecutar para el arranque del sistema y el archivo *last_id.txt* con el último id al que se ha respondido.

Tabla 67. Contenido del archivo adjunto.

14.3 Actas de reuniones

Se muestran a continuación las actas referentes a las reuniones realizadas con el tutor durante la realización del proyecto.

Reunión	Fecha	Medio	Contenido
Reunión 1	03/03/2022	Microsoft Teams	Reunión Inicial
Reunión 2	31/03/2022	Microsoft Teams	Actualización en la investigación previa
Reunión 3	12/05/2022	Microsoft Teams	Actualización en la investigación previa y documentación inicial
Reunión 4	09/06/2022	Microsoft Teams	Actualización en la investigación previa, planificación y presupuesto
Reunión 5	23/06/2022	Microsoft Teams	Actualización en la investigación previa, planificación y presupuesto
Reunión 6	15/09/2022	Microsoft Teams	Actualización en la búsqueda de Datasets
Reunión 7	22/09/2022	Microsoft Teams	Actualización y problemas encontrados durante la generación y comparativa de modelos
Reunión 8	06/10/2022	Microsoft Teams	Actualización y problemas encontrados durante la generación y comparativa de modelos
Reunión 9	20/10/2022	Microsoft Teams	Actualización y problemas encontrados durante la generación y comparativa de modelos
Reunión 10	27/10/2022	Microsoft Teams	Actualización y problemas encontrados durante la generación y comparativa de modelos
Reunión 11	03/11/2022	Microsoft Teams	Actualización y problemas encontrados durante la generación y comparativa de modelos
Reunión 12	10/11/2022	Microsoft Teams	Actualización en el desarrollo del sistema y problemas encontrados
Reunión 13	17/11/2022	Microsoft Teams	Actualización en el desarrollo del sistema y problemas encontrados
Reunión 14	15/12/2022	Microsoft Teams	Actualización en el desarrollo del sistema y problemas encontrados

Reunión 15	22/12/2022	Microsoft Teams	Actualización en el desarrollo del sistema y problemas encontrados
Reunión 16	12/01/2023	Microsoft Teams	Revisiones de Documentación
Reunión 17	19/01/2023	Microsoft Teams	Revisiones de Documentación
Reunión 18	26/01/2023	Microsoft Teams	Revisiones de Documentación
Reunión 19	02/02/2023	Microsoft Teams	Reunión Final

Tabla 68. Actas de reuniones.