Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Procesamiento del Lenguaje Natural

Natural language processing

La Laguna, 10 de junio de 2021

D. **José Luis González Ávila**, con N.I.F. 78.677.390-W profesor Asociado adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

Dª. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

"Procesamiento del lenguaje natural"

ha sido realizada bajo su dirección por D. **Jordi Hernández Hernández**,

con N.I.F. 42.418.167-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2021

Agradecimientos

A mis tutores José Luis González Ávila e Isabel Sánchez Berriel por toda la ayuda e ideas aportadas para que el proyecto cumpliera su función, y sin los cuales me hubieran surgido muchas dificultades.

A mi círculo de confianza, por no dudar de mí, y por darme su tiempo para ayudarme en todo lo que necesitaba, incluso sin pedirlo.

Licencia

Resumen

Las redes sociales son plataformas que forman parte de nuestras vidas, nos informan, entretienen, y nos permiten relacionarnos entre nosotros. A parte de su uso general, poseen unas interfaces llamadas API que permiten a los desarrolladores obtener y enviar datos de manera bidireccional con sus bases de datos.

Entre estas redes está Twitter, cuya interfaz es muy utilizada, tanto para obtener información relevante como para enviar datos en forma de respuesta, tweets, u otros métodos.

El procesamiento del lenguaje natural es una rama de la inteligencia artificial la cual pretende replicar la facultad del lenguaje humano. Esta tecnología supone una evolución enorme, debido a que nos permite hacer una amplia cantidad de tareas en cualquier ámbito en el que intervenga el lenguaje humano, como es nuestro caso, empleándolo para obtener información relevante de tweets y relacionarlos entre sí.

El objetivo de este trabajo es, mediante el análisis de nuestro timeline en Twitter, obtener el texto con mayor relación en nuestra base de datos, pudiendo responder de manera autónoma a los mensajes manteniendo una cierta coherencia.

***Palabras clave**: Twitter, Procesamiento del Lenguaje Natural, Inteligencia Artificial.*

Abstract

Social networks are platforms that are part of our lives, they inform us, entertain us, and allow us to interact with each other. Apart from their general use, they have interfaces called APIs that allow developers to obtain and send data in a bidirectional way with their databases.

Among these networks is Twitter, whose interface is widely used, both to obtain relevant information and to send data in the form of responses, tweets, or other methods.

Natural language processing is a branch of artificial intelligence which aims to replicate the faculty of human language. This technology represents a huge evolution, because it allows us to do many tasks in any field in which human language intervenes, as is our case, using it to obtain relevant information from tweets and relate them to each other.

The objective of this work is, through the analysis of our timeline on Twitter, to obtain the text with the greatest relation in our database, being able to respond autonomously to the messages while maintaining a certain coherence.

***Keywords**: Twitter, Natural Language Processing, Artificial Intelligence.*

Índice general

Índice de figuras

Capítulo 1
Introducción

Ya es tópico decir que el procesamiento del lenguaje natural está presente en nuestra vida cotidiana, su uso por parte de las organizaciones y personas ha supuesto un gran paso en la sociedad. Este tipo de procesamiento permite a las máquinas reconocer el lenguaje humano y realizar infinidad de tareas, como es en nuestro caso la obtención de información mediante las redes sociales.

1.1 Objetivos

El objetivo de este proyecto es crear un programa capaz de contestar, mediante uso del procesamiento del lenguaje, a tweets de manera autónoma en la red social de Twitter.

Su funcionamiento se basa en captar mensajes desde el timeline de Twitter, para posteriormente aplicarle técnicas de inteligencia artificial que permita encontrar el texto con mayor relación de nuestra base de datos y responder de forma automática manteniendo cierto nivel de coherencia.

1.2 Alcance

Las distintas fases que se llevarán a cabo en este proyecto implican las siguientes tareas:

- Integración con una aplicación de interfaces de la plataforma de Twitter
- Obtención de un corpus de tweets relacionados con la política.
- Creación de una base de datos NoSQL para almacenar el contenido del corpus.
- Aplicación de técnicas de procesamiento del lenguaje sobre los textos obtenidos para poder establecer una relación semántica entre ellos.
- Crear una escucha activa en la plataforma de Twitter.

- Por cada tweet que se recibe, aplicarle procesamiento del lenguaje para conseguir el documento con mayor relación de nuestra base de datos.
- Responder al tweet con contenido procedente del documento seleccionado.
- Evaluación de resultados.

1.3 Antecedentes

En este apartado presentamos ejemplos que servirán de muestra del potencial de la extracción de conocimiento en las redes sociales.

En primer lugar tenemos el caso de la solución presentada en el artículo 'A review on political analysis and social media' [1] en el que se describe la aplicación del procesamiento del lenguaje natural en redes sociales para conocer la intención de los votantes de una zona. Otro caso de uso lo encontramos en el proyecto de minería de opiniones acerca de los hoteles, que se basa en la clasificación de las opiniones, en si son positivas o negativas, que se presenta en el artículo: 'Minería de Opiniones basado en la adaptación al español de ANEW sobre opiniones acerca de hoteles' [2]. Por último, se menciona el trabajo fin de grado de Noel Padrón Díaz: 'Automatización de respuestas a tweets mediante PLN' [3] en el que, mediante unas keywords presentes en cada documento de una base de datos, es capaz de responder a tweets cuyo mensaje coincida con esas palabras, recomendando un artículo/noticia.

1.4 Destinatarios

Nos encontramos ante un proyecto cuya finalidad es la de crear una herramienta que permita a cualquier organización establecer, por ejemplo, conversaciones con personas cuyos tweets estén relacionados con un tema a su elección, y de esta manera dar a conocer su entidad recomendando en su respuesta datos, documentos, artículos, etc.

Capítulo 2
Análisis de tecnologías

En este capítulo se exponen los aspectos y tecnologías que se han analizado en el desarrollo del trabajo.

2.1 Twitter

Twitter es una red social, una de las primeras, que se presentó en marzo de 2006, pero su primera aparición surge alrededor de 2004, como un proyecto de investigación de la empresa Odeo. Esta plataforma permite a los usuarios leer y escribir textos que reciben el nombre de tweets, y tienen 140 caracteres de longitud máxima. Este tamaño no fue elegido de manera arbitraria, sino que se estableció debido a que los mensajes se enviaban a través del servicio SMS el cual tenía este límite establecido por los proveedores. Tras crear una forma de envío distinta de los mensajes se aumentó su tamaño máximo a 240 caracteres, incrementando de esta manera la fama y uso de la red social.

Un dato curioso es que su nombre inicial fue de twttr, obviando las vocales para seguir la tendencia de nombres que se empleaba en esa época. Además, su nombre fue elegido por cómo se pronunciaba en inglés el sonido que generaban los pájaros [‘tweett’]

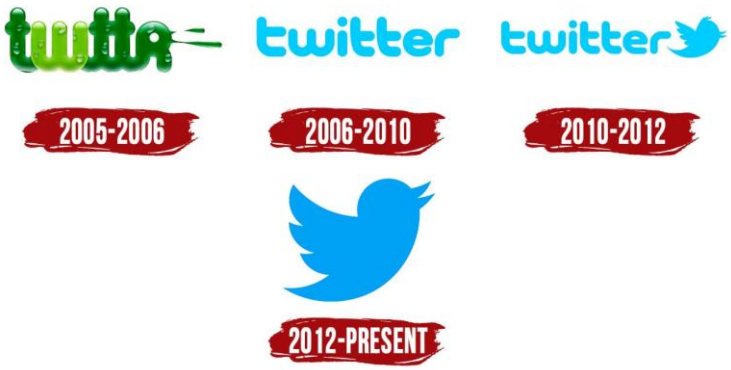

Figura 1: Evolución del logo de Twitter

Para entender un poco más la plataforma vamos a definir una serie de términos que existen en Twitter:

- Mensaje directo: Consiste en enviar un mensaje que solo verá el usuario al cual se lo has enviado. Es el chat común que existe en la gran mayoría de redes sociales.
- Favorito: Es la acción de dar 'me gusta' a una publicación. Esto permite al propio usuario poder verlo posteriormente en su sección de 'Me Gusta', además de aumentar la visibilidad del tweet en cuestión, ya que el resto de los usuarios que sigan a este usuario verán en su timeline el contenido al que le da me gusta.
- Follow: Comenzar a seguir a una persona para poder comunicarte con ella, ver que contenido sube, y que sus tweets aparezcan en tu timeline.
- Seguidores: Número de personas que han hecho follow a una persona. Se puede ver en la página de perfil de ese usuario. A mayor número de seguidores más fama y, por tanto, influencia tiene la cuenta.
- Siguiendo: Cantidad de personas que sigue un usuario. Al igual que los seguidores, se puede ver en su perfil, pero en este caso no afecta a la influencia del usuario. Las personas que sigues aparecerán en tu timeline.
- Hashtag: Emplean el elemento '#', y se usa para escribir tweets que sigan un tema concreto de la plataforma. Todos los mensajes que tengan esta etiqueta aparecerán englobados en el mismo tema.
- Mención: Cuando se escribe un tweet existe la posibilidad de escribir el usuario de una persona concreta en él, de esta forma a la persona mencionada le aparecerá una notificación con el contenido en el cual se le ha incluido.
- Reply: Responder a una persona en concreto. Cuando se escribe un tweet, existe la posibilidad de responder a ese tweet mencionando a una persona mediante el símbolo '@' para que se notifique a dicho usuario de la respuesta.
- Retweet: Con esta acción publicas el tweet de otra persona en tu propio perfil. Es una manera efectiva de hacer llegar a tus seguidores el contenido de otra

persona que quizás no conozcan y te haya parecido interesante.

- Timeline: También conocido como biografía, es la página principal de Twitter, en ella se muestran todas las interacciones que tienen las personas a las que sigues, sus publicaciones, los contenidos a los que han dado me gusta o que han hecho retweet.
- Tweet: Es el contenido del mensaje que escribes para que se publique en tu perfil y sea visible en el timeline de los usuarios que te siguen.
- Trending Topics: Son los temas más famosos del momento, pueden incluir a personas relevantes, hashtags, temas, etc.

2.2 Bots

Más allá del uso general que tiene esta plataforma, y en relación con el contenido de este proyecto, nos encontramos los bots. Se definen como una especie de tecnología que simula el uso de la red social por parte de una persona. Son creados con fines determinados que van más allá del uso general que se le da a esta red. Estos robots son creados tanto por personas individuales como por organizaciones completas, los distintos funcionamientos que estos poseen pueden ser clasificados en varios tipos:

- Testing Bot: son creados por la propia empresa que posee un producto con el fin de probar las funciones e interacción con el mismo. Esta herramienta les permite tener un ambiente controlado en el que testear sus productos.
- Following bot: tienen una simple función, la cual es seguir a otros usuarios. Este tipo de robot se emplea para aumentar la cantidad de seguidores de una cuenta o para hacer crecer la propia cuenta siguiendo a un gran número de personas. Habitualmente se emplean como negocio en la compra y venta de seguidores dentro de la plataforma.
- Traffic bot: se usan para generar tráfico dentro de las plataformas, páginas web, etc. Y dentro de la red social de Twitter se emplean para acciones como dar me gusta o retwittear. Es una manera de dar visión a un contenido deseado.
- Sending bot: su objetivo es intentar, mediante el empleo de tweet y retweet, qué un cierto tema o hashtag se convierta en trending topic dentro de la

plataforma.

- Crisis bot: están programados para seguir un tema o una cuenta determinada y defenderlo o atacar en función de un objetivo determinado. En ocasiones son empleados para provocar una imagen pública determinada sobre una marca.

En nuestro caso el objetivo a seguir es un poco distinto al de los bots estándar. Nuestro programa será capaz de obtener de una base de datos propia los documentos presentes en ella, para de esta manera poder responder a los Tweets que escriban los usuarios de nuestro Timeline con un mensaje que tenga relación con lo que se está hablando en el tweet.

2.3 Procesamiento del lenguaje natural

El procesamiento de lenguaje natural da la posibilidad de establecer una comunicación entre una persona y una máquina a través de lenguajes naturales como el español. El procesamiento del lenguaje es definido como una rama de la inteligencia artificial la cual pretende, mediante entrenamiento y distintos ámbitos de la programación, replicar la facultad del lenguaje humano. Y su uso supone una evolución enorme, debido a que nos permite hacer una amplia cantidad de tareas con una finalidad concreta, como puede ser la traducción, establecer una conversación, y obtener contenidos de las redes sociales entre otras tantas posibilidades. Dicho de una manera más general, este procesamiento se puede aplicar con cualquier faceta en la que intervenga el lenguaje humano.

Los ordenadores son capaces de analizar la lengua humana, independientemente del idioma en el que estemos hablando. Pero el lenguaje humano se presenta complejo para quienes no dominan su código. Debido al empleo de los tiempos verbales, las distintas maneras de comunicar una misma palabra, y la existencia de jergas, las máquinas requieren hacer uso de estos programas para poder comprender en su propia lengua el idioma de las personas.

Es evidente pensar que los ordenadores sólo entienden bytes y dígitos binarios. Por lo tanto, para entender el lenguaje humano es necesario darle a la máquina un modelo lingüístico con el cual pueda ir más allá de estas limitaciones. Los modelos para el procesamiento de lenguaje natural que existen son de dos tipos:

- Lógicos: estos modelos están basados en determinadas formas gramaticales, de manera que los lingüistas especializados definen diferentes patrones estructurales de la lingüística para que las máquinas puedan identificarlo. Si a este concepto le añadimos la información existente en los distintos diccionarios computacionales damos la posibilidad de definir modelos que son capaces de resolver determinadas tareas como pueden ser buscar información, traducir, resumir etc.
- Probabilístico: A diferencia del modelo lógico en este no se valoran las reglas gramaticales en ningún momento. Se basa en la recolección por parte de los lingüistas de una gran cantidad de datos que se emplearán como foco principal en el análisis. A partir de estos datos se analizan y localizan la frecuencia de distintos elementos lingüísticos como pueden ser letras o palabras. A partir de este análisis se puede predecir la siguiente unidad que aparecerá en el texto basándonos en la probabilidad de esta. En el ámbito de la inteligencia artificial esto se conoce como aprendizaje automático.

Según la finalidad que tenga el programa de procesamiento utilizará un análisis diferente, pero en general existen 4 tipos:

- Análisis morfológico: es el análisis más interno que se le realiza a las palabras, y su finalidad es obtener su significado y categoría.
- Análisis sintáctico: analiza el texto para obtener la estructura de cada una de las frases que lo componen.
- Análisis semántico: su finalidad es conseguir una correcta deducción de las frases.
- Análisis pragmático: pretende obtener una correcta conclusión del contexto.

El empleo del procesamiento del lenguaje es muy amplio, algunos ejemplos de uso en la interpretación, obtención de información y análisis morfológico son:

Para traducir documentos de manera autónoma, permitiendo a los traductores aumentar su eficiencia, ya que estaría trabajando sobre documentos previamente procesados por la máquina.

En las redes sociales: permite a las empresas aumentar su ritmo en la lectura y obtención de información relevante para su organización, ya que el programa es capaz de comunicarle en qué mensajes hay documentación relacionada con sus intereses.

Definiendo una serie de categorías, es capaz de separar textos clasificándolos dentro de estas.

2.4 Base de datos no relacional

Las bases de datos existen con la finalidad de almacenar información en ella. Toda aplicación que requiera del almacenaje de datos recurrirá al empleo de estas.

Las primeras que existieron son las llamadas bases de datos relacionales, como mariadb. Son un conjunto de tablas que guardan atributos y sus valores. El nombre que reciben estas bases se debe a que las distintas tablas pueden relacionarse entre sí.

Durante una larga época fueron la solución perfecta para almacenar información, pero a medida que pasaba el tiempo, la necesidad de guardar gran cantidad de datos era cada vez mayor, y a su vez, se encontraba que el rendimiento de estas bases se veía reducido como consecuencia de un mayor almacenaje de información.

Frente a esta deficiencia surgen las bases de datos NoSQL (not only sql). La ventaja de estos sistemas de almacenamiento radica en que no emplean el esquema de tablas y relaciones, sino que se almacena la información de la misma manera que piensan los programadores, como si fueran objetos.

Originalmente el término NoSQL se empleó en el año 1998 para hacer referencia a una base de datos open source que destacaba por su ligereza, y por no emplear una interfaz sql.

La mayor velocidad de estos sistemas se basa en que, al no tener relaciones entre sí, no se realizan consultas ‘join’, que son instrucciones que van buscando un valor de entre las tablas que se encuentran relacionadas, ralentizando notablemente las búsquedas cuando estas tablas eran suficientemente grandes. Por otra parte, las bases SQL siguen una estructura estricta, con campos fijos, de manera que si se quiere añadir algún atributo más es necesario modificar la estructura completa de la base. En las NoSQL la estructura es libre, al tratarse la información sin unos esquemas estrictos,

cada conjunto de datos puede tener sus propios atributos, no es necesario que todos tengan la misma cantidad de campos.

En bases de datos no relacionales, como es el caso de MongoDB, se emplea el lenguaje JSON para guardar los datos. JSON proviene de 'javascript native object' y guarda la información de manera similar a la que se hace en un documento.

```
{
  "firstName": "Jonathan",
  "lastName": "Freeman",
  "loginCount": 4,
  "isWriter": true,
  "worksWith": ["Spantree Technology Group", "InfoWorld"],
  "pets": [
    {
      "name": "Lilly",
      "type": "Raccoon"
    }
  ]
}
```

Figura 2: Documento JSON

Dentro de las NoSQL encontramos diversas formas de abarcar un problema concreto. Las podemos dividir en:

- Tipo clave-valor: Son el tipo de datos más simple, por cada dato a almacenar guardan una clave que lo identifica y su valor. Su velocidad de consulta es muy eficaz tanto para lectura como en escrituras. Permite escalar horizontalmente de manera muy sencilla, ya que pueden ser distribuidas en servidores en función del numero de la clave (Ejemplo: Si la clave está entre 0-100 va al server1, si está entre 100-200 va al server2).
- Bases en columnas: La información se almacena manteniendo un parecido a las tablas de las bases relacionales, pero en este caso se almacenan por columnas independientes en lugar de a lo largo de una única fila. Es un método muy efectivo si lo que se quiere hacer es leer la mayor parte del tiempo.
- Orientadas a documentos: Estas bases presentan una gran ventaja, dado que mantienen las prestaciones de las bases relacionales al guardarles el key de manera que la base de datos pueda comprenderla, pero definiéndose como de tipo scheme-less. Se siguen guardando dos campos, una para la clave y otra

para el valor, pero en este caso la clave puede ser interpretada por la base de datos. Este tipo de bases permite incluso crear relaciones.

- Orientada a grafo: En este caso la información se almacena en los nodos del grafo, mientras que las relaciones son los enlaces entre los nodos. Con este tipo de bases se les da una mayor importancia a las relaciones y no solo a la información. Estas bases permiten una escalabilidad horizontal sencilla, y las consultas con respecto a las relaciones son más rápidas que en las bases relacionales.

Capítulo 3
Herramientas utilizadas

En el capítulo anterior se han introducido las tecnologías en las que se basará este proyecto. Ahora procederemos a definir una seria de herramientas que nos serán de utilidad para llevar a cabo el desarrollo del programa de una manera más sencilla y eficaz.

3.1 Python

Python es un lenguaje de programación dinámico que se ha convertido con el tiempo es uno de los más utilizados por los programadores. Es un lenguaje interpretado de alto nivel, esto significa que posee una estructura legible para los humanos, tanto sintáctica como semánticamente. Al ser un lenguaje interpretado no necesita de un compilador para ejecutarse, en su lugar emplea un intérprete que ejecuta el programa directamente.

Este lenguaje surgió con la idea de ser un sucesor del lenguaje ABC que usaba su creador, Guido Van Rossum, en su trabajo en un centro de investigación holandés de carácter oficial. En un principio sería un lenguaje de scripting, pero con el paso del tiempo se ha convertido en un lenguaje de propósito general.

Python es un lenguaje multiparadigma, ya que soporta programación orientada a objetos, programación imperativa y, en menor medida, programación funcional. Otra característica es que es de tipado dinámico, esto significa que no es necesario declarar el tipo de datos de las variables, ya que el lenguaje le autoasigna un tipo concreto según el valor que reciba.

Existen múltiples entornos de programación para Python, como por ejemplo IPython+Jupiter, que permite crear cuadernos de notas de Jypiter en los que se ejecuta Python, esto permite la colaboración entre proyectos, y emplear computación en la nube como la que ofrece Google Collab, el cual hemos usado en este proyecto.

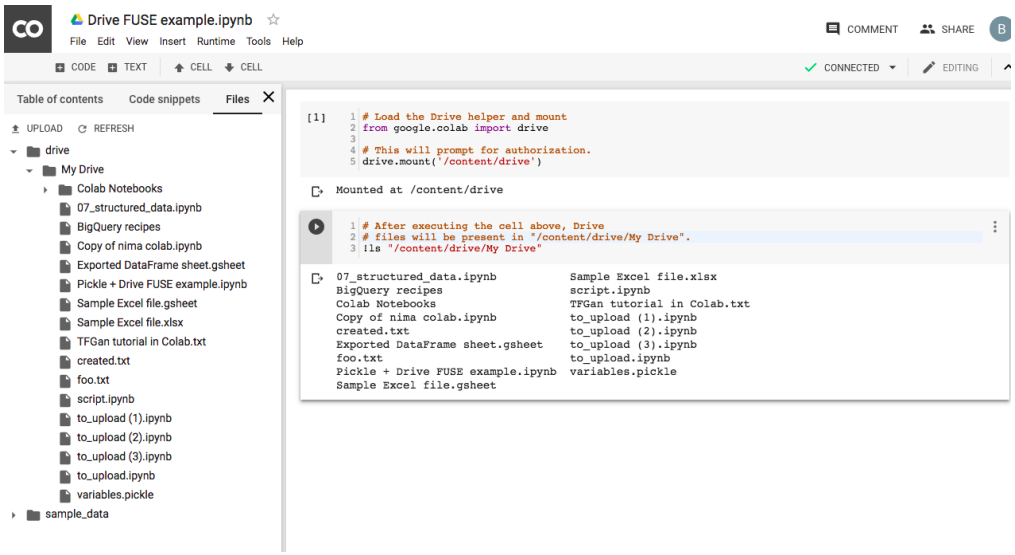

Figura 3: Entorno de Google Collab

Python posee una gran cantidad de bibliotecas y programas para desarrollar funcionalidades. Es un lenguaje con una gran cantidad de recursos en cuanto a procesamiento del lenguaje natural se refiere, y una de ellas es NTLK.

3.1.1 NLTK

NTLK son las siglas de kit de herramientas de lenguaje natural (‘Natural Languague Toolkit’). Consiste en una serie de bibliotecas y programas creados para el lenguaje de Python, que permiten hacer procesamiento del lenguaje general.

Algunos de los recursos usados en este proyecto son:

- Tokenizar texto: Permite separar el contenido del texto en palabras, etiquetadas como tokens.
- Eliminar Stopwords: Ntlk tiene un diccionario en la mayoría de los idiomas que contiene palabras sin significado que es necesario eliminar de los textos para reducir el ruido. Son llamadas ‘Stopwords’.
- Lematizar: Para cada palabra dada, obtiene el lema de esta. Dicho de otra manera, cuando una palabra esta flexionada (en plural, en femenino, en algún tiempo verbal, etc.) el programa obtiene la forma general que representa el resto de las flexiones.

3.2 Tweepy

Tweepy es una librería para el lenguaje de programación Python que permite el acceso a la información de la API de Twitter a través de la propia cuenta del programador. Las ventajas que presenta esta herramienta radican en la facilidad de uso que tiene para obtener las funciones deseadas, ya que sin este intermediario deberíamos tener en cuenta muchas peticiones de bajo nivel relacionadas con solicitudes http de la página, limitaciones de velocidad, accesos a la aplicación y demás condiciones tediosas en las cuales no queremos pensar a la hora de hacer un programa de forma sencilla.

Para acceder a la información y las funcionalidades es necesario hacerse una cuenta como desarrollador en la red social. Twitter permite hacer esto a través de un formulario en su página de desarrollo la cual te provee de unas claves de consumidor, únicas para cada usuario, que tendrán que introducir a la hora de usar la interfaz.

Tweepy presenta una diversidad de métodos que van más allá de únicamente obtener la información del timeline. Se han clasificado según su funcionalidad:

- Métodos de línea de tiempo: Devuelve el contenido de tu timeline. Por defecto te retorna los 20 primeros elementos.
- Métodos de estado: Este método publica un estado en el perfil personal del usuario con el que estés conectado a la api.
- Métodos de usuario: Obtienes la información de un usuario concreto. Dentro del objeto devuelto tienes datos como el nombre completo, username y número de seguidores.
- Métodos de mensajes directos: Envía un mensaje a un usuario concreto a través del chat privado entre usuarios existente en la red social.
- Métodos de amistad: Sería la acción similar a “comenzar a seguir” o “dejar de seguir” a una persona determinada.
- Métodos favoritos: Este método da “me gusta” a un tweet mediante el identificador del mensaje (ID). Si previamente estaba como favorito esta acción lo quitará.
- Métodos de bloque: Con esta función es posible bloquear a un usuario de tu cuenta, o desbloquearlo en caso de que ya lo estuviera.
- Métodos de búsqueda: Permite buscar un tweet o un conjunto de ellos que coincidan con algún parámetro establecido. Atributos como ‘q’ buscan

mensajes que contengan el texto definido en esa variable, o 'lang' que define el idioma del tweet.

- Métodos de tendencias: Mediante una clasificación de zonas geográficas a través de números, esta funcionalidad devuelve las tendencias actuales en esa zona.
- Métodos geográficos: Devuelve de manera específica, incluyendo las coordenadas si es posible, de un lugar concreto especificado mediante su id.

3.3 Gensim

Dentro del procesamiento del lenguaje natural, Gensim es una librería de código abierto que existe para el lenguaje de programación de Python. Esta herramienta de modelado permite, mediante un documento de entrada, transformarlo en un vector semántico de palabras.

Para entender mejor esta explicación definiremos una serie de conceptos:

- Documento: Un documento no es más que un texto que puede abarcar una longitud que va desde una frase hasta un libro, pasando por un simple párrafo.
- Corpus: Es la entrada estándar de procesado de gensim, definido como un conjunto de objetos de tipo documento. Un ejemplo de esto podría ser un conjunto de libros, donde los libros son los documentos y el conjunto conformaría el corpus. Estas colecciones son usadas por gensim para entrenar los modelos en un primer paso, y posteriormente reconocer temas similares en las pruebas.
- Vector: Cuando gensim trabaja sobre un documento, lo procesa, y lo representa como una estructura que contiene para cada texto una lista de palabras con el índice de la palabra y su frecuencia.

```
(1, 0.0), (2, 2.0), (3, 5.0)
```

Figura 4: Vector con índice de la palabra y su frecuencia

El origen de gensim es un poco diferente a su función actual. En sus inicios gensim estaba formado por una serie de scripts escritos en python para formar parte de una librería matemática. Estos scripts ordenaban, dado un artículo matemático, la lista de los

artículos más similares a este.

A medida que avanzaba su desarrollo, aumentaba enormemente su eficiencia y escalabilidad, esto hizo que con el paso del tiempo se convirtiera en un software libre de modelado semántico sobre texto plano.

Gensim implementa, entre otros, los modelos del tipo word embeddings, WordtoVec y DoctoVec mediante el aprendizaje a través de redes neuronales de vectores de palabras en un espacio semántico. En este caso nos centraremos en DoctoVec.

Este modelo transforma los documentos de un corpus de entrada en un conjunto de vectores numéricos en un espacio multidimensional en el que la similitud numérica de los vectores refleja la similitud del contenido semántico de los documentos. Este modelo del lenguaje o espacio semántico se obtiene aplicando redes neuronales que cuyo input en la primera iteración se obtiene a partir de las frecuencias de las palabras en cada documento y en el corpus. Este vector permite al ordenador trabajar de manera eficiente. Para conseguirlo se siguen una serie de pasos que incluyen varias funciones presentes en la herramienta.

Las funciones principales son:

- Simple_preprocess: El preprocesado de gensim transforma a minúscula las palabras de un texto pasado como parámetro. Además, elimina los signos de puntuación, así como las stopwords del idioma en el que estemos trabajando.
- Build_vocab: Crea un vocabulario compuesto por las palabras presentes en el corpus.
- Train: función que llama al modelo concreto definido de doc2vec para entrenarlo con texto procesado previamente.
- Infer_vector: genera un vector con una serie de palabras que recibe como parámetro. Este elemento se emplea para buscar el parecido con otros textos mediante la similitud del coseno.
- Most_similar: Esta función coge el vector que se ha generado en la función ‘infer_vector’ y busca el vector más similar procedente de los textos que han sido entrenados previamente.

1. En primer lugar, el fichero corpus es leído por una función que abre el archivo y le realiza un preprocesado simple.

2. Posterior a esto se define el modelo Doc2Vec con sus atributos: tamaño del vector, frecuencia mínima (para eliminar palabras que aparecen pocas veces), y el número de veces que se repetirá el entrenamiento de la red neuronal.
3. Generamos un vocabulario constituido por las palabras que entrenaremos.
4. Una vez tenemos el vocabulario definido, lo llamamos mediante la función 'train' para que entrene con el documento procesado.

En este momento ya es posible evaluar el modelo, con documentos que no hayan sido usados para entrenar al modelo, generamos un vector mediante la función 'infer_vector' y le pedimos al modelo que nos devuelva el documento más similar con respecto a los textos entrenados previamente.

De esta manera podemos relacionar textos a través de su representación mediante vectores numéricos.

3.4 Mongo DB

Mongo DB es una base de datos de tipo NoSQL. Este tipo de bases es la mejor elección cuando no se tiene un concepto predefinido de datos que se van a guardar.

Según la página oficial, es la base de datos no relacional más utilizada en el mundo. El principal motivo de este posicionamiento es que permite el almacenamiento masivo de datos sin pérdida de eficiencia.

Esta base de datos emplea almacenamiento orientado a documentos. Para entender mejor su funcionamiento la compararemos con el esquema de una base de datos relacional.

En una base de datos SQL la estructura es una tabla que contiene un atributo por columna, y una fila por cada elemento que se almacene con dichos atributos. En caso de que se quiera almacenar un nuevo atributo es necesario modificar la tabla para añadir una nueva columna. En Mongo DB lo que se consideran tablas se llaman colecciones, a su vez las colecciones contienen un conjunto de documentos, estos serían lo que en las bases de datos relacionales se denominan filas. Dentro de los documentos se encuentran los atributos, cada atributo tiene una clave con el nombre del atributo y un valor. Si se quisiera añadir un atributo nuevo en alguno de los documentos, únicamente habría que escribir su clave-valor dentro del documento. Esta flexibilidad que presenta Mongo DB

nos da la posibilidad de tener documentos con tamaño variable, y escapar de la rigidez de tener que modificar una base de datos completa para añadir un atributo que quizás únicamente tiene uno de los elementos de la tabla.

Para llevar a cabo el almacenamiento de estos datos se usa el formato JSON (Java Script Object Notation). Un JSON es un documento compuesto por pares de datos clave-valor. Este documento no tiene tamaño fijo ni reglas concretas. Los tipos de datos que admiten los valores asociados a las claves en el formato JSON son los siguientes:

- String: una cadena de caracteres.
- Número.
- Objeto: el valor de una clave puede ser un nuevo conjunto de datos del tipo clave-valor. Se especifican mediante {}
- Arrays: el valor está compuesto por un subconjunto de valores. Se especifican mediante []
- Booleano: true o false
- Null: es similar a dejar un campo en blanco.

Las principales ventajas por las que se usan estas bases de datos en lugar de una relacional son:

- Al almacenar todos los datos que pertenecen a un objeto en un mismo documento es más fácil y rápido acceder a ellos.
- No es necesario unir diferentes tablas para obtener valores relacionados con un objeto, ya que están todos dentro del documento.
- Aporta una mayor flexibilidad a la hora de crear relaciones y con el diseño del schema.

3.5 Visual Studio Code

Presentado en 2015 por Microsoft, Visual Studio Code es un editor de código fuente gratuito y de código abierto. Este editor fue lanzado bajo la licencia Mit y se puede encontrar como repositorio en github. Esta herramienta se puede usar en Windows, Mac y Linux.

Electron es el framework elegido en el que se basó la creación de Visual Studio

Code. Este framework surgió para impulsar como aplicaciones de escritorio a chromium y nodejs.

VSC permite programar en una infinidad de lenguajes de programaciones, esto junto a la personalización de los temas es uno de los tantos motivos por lo que es tan utilizado.

Junto a la compatibilidad con los diferentes lenguajes, esta herramienta contiene una serie de características que están disponibles para algunos lenguajes de forma nativa. Algunas de ellas son:

- Resaltado de sintaxis.
- Snippets: Reciben este nombre algunas pequeñas partes de código que pueden ser reutilizadas en módulos más grandes.
- Autocompletado de código: mediante Inteligencia Artificial, VSC es capaz de sugerir que función/palabra están escribiendo y sugerírtela para no tener que escribirla completamente.
- Refactorización: En principio esta característica únicamente está disponible para C# y Typescript.
- Depuración.

Aunque no todas las características están disponibles para la mayoría de los lenguajes, una de las grandes ventajas que presenta este editor frente a sus competidores es la existencia de extensiones. Un Marketplace en el cual existen diversidad de herramientas y recursos que permiten agilizar el desarrollo de programas en el lenguaje de programación que desees. Esta característica le ha permitido posicionarse delante de sus predecesores, como pueden ser Sublime, Vim, etc.

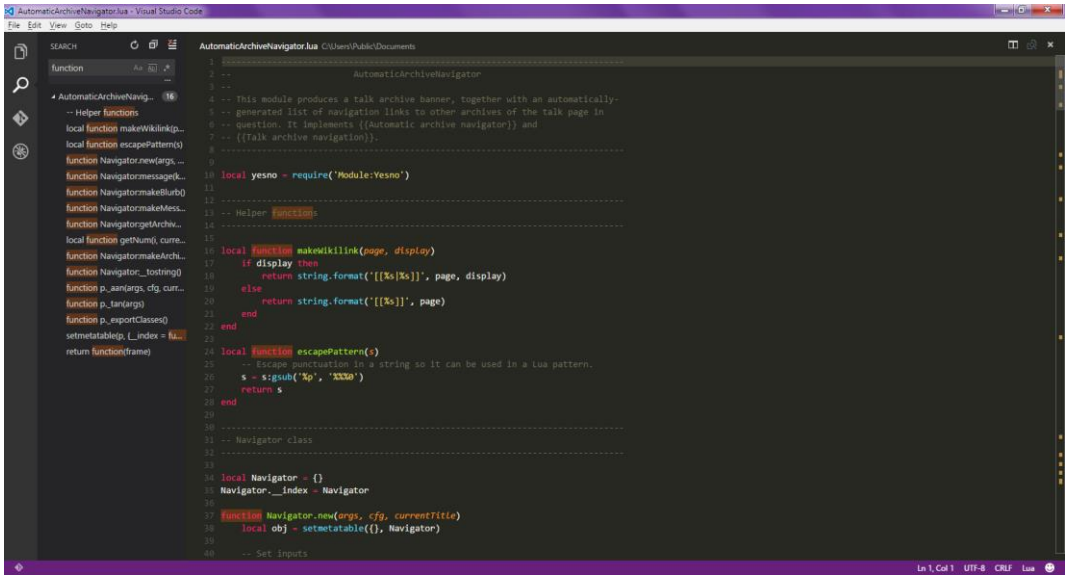

Figura 5: Vista general de VSC

3.6 Git

Git es un sistema de control de versiones para tu proyecto, es gratuito y de código abierto. Git permite guardar los cambios que se van haciendo en tu código, es decir, en lugar de guardar una versión completa del código con cada cambio que hagas, Git almacena únicamente el cambio que se produce en cada versión, permitiendo volver en cualquier momento a la versión deseada, independientemente de cuántos cambios se hayan hecho posteriormente.

Otra característica de git es que permite cooperar en proyectos, mediante la creación de ramas (copias del proyecto) varios usuarios pueden trabajar en funcionalidades diferentes sin modificar los archivos del otro, para posteriormente fusionarlos en uno.

3.7 Github

Es un servicio para el control de versiones alojado en la web que permite almacenar tu código en la nube. Es considerado como la red social de los desarrolladores. Te permite ver el código de otras personas, e incluso colaborar con ellos, identificando fallos y proponiendo soluciones. Github te permite almacenar y descargar el código de un programa tantas veces como quieras. El contenido de un programa se almacena en repositorios (lugar de la nube donde se almacenan los archivos de un código escrito en algún lenguaje de programación).

3.8 Source Tree

Esta herramienta hace mucho más fácil comunicarte con tu repositorio, permitiendote centrarte en la codificación de tu programa. Es una interfaz visual que simplifica la forma de interactuar con git, permitiendo ver los cambios realizados y controlar qué hacer con tus repositorios sin tener que escribir ningún comando. Esta interfaz te permite hacer las mismas acciones que en un terminal, pero facilitando la tarea mediante botones en una app bastante intuitiva, ahorrándote el tiempo de pensar qué comando tienes que escribir.

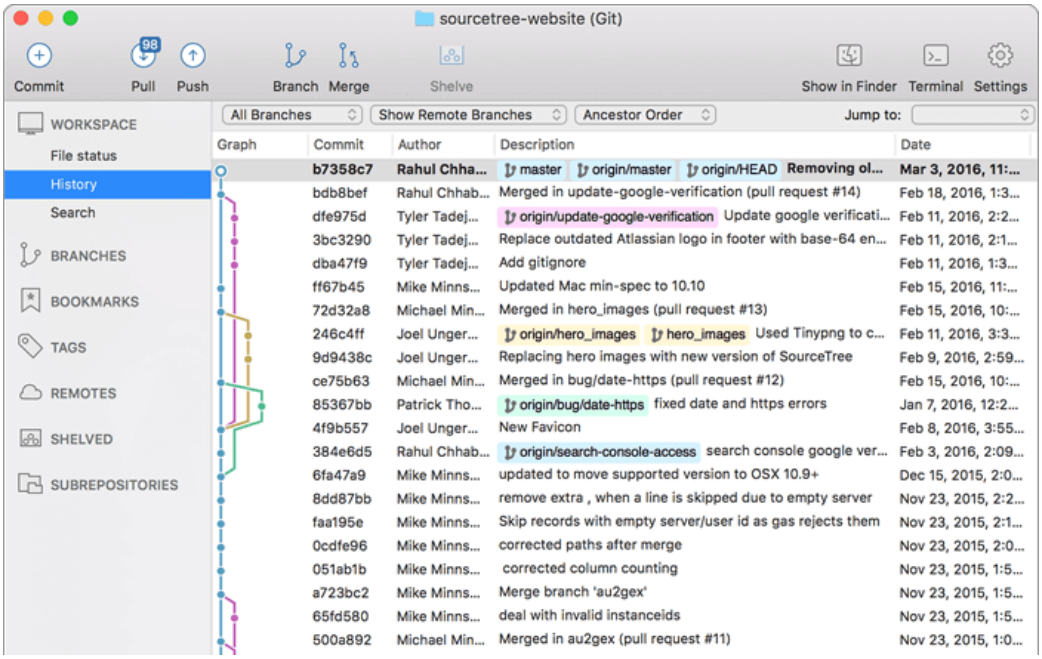

Figura 6: Panel principal de Source Tree

Capítulo 4
Desarrollo

4.1 Metodología y fases del proyecto

La elaboración del proyecto se ha dividido en dos programas principales. El primer programa está encargado de obtener, mediante la api de Twitter, el contenido de los tweets que se encuentran en el corpus que se ha seleccionado. Para cada mensaje adquirido se le aplica un preprocesado, para posteriormente almacenarlo en la base de datos. Tras haber realizado este paso con todo el corpus, se analizan los documentos para poder generar los tópicos con los que evaluaremos la fiabilidad del modelo.

En cuanto al segundo programa, este se centra en la parte del modelo de aprendizaje. Se crea un modelo doc2vec con los parámetros necesarios. Posteriormente se genera un corpus de tweets distintos a los que se emplean para el entrenamiento, obtenidos a través del timeline de Twitter. Al igual que en el primer programa, se le aplican funciones de preprocesado para eliminar el ruido. Con este paso finalizado comienza el entrenamiento del modelo, que llevará un intervalo de tiempo concreto en función de los valores que se le hayan dado al crearlo. Una vez finalice genera, para cada elemento del corpus de prueba, un documento similar para poder responder al tweet en cuestión.

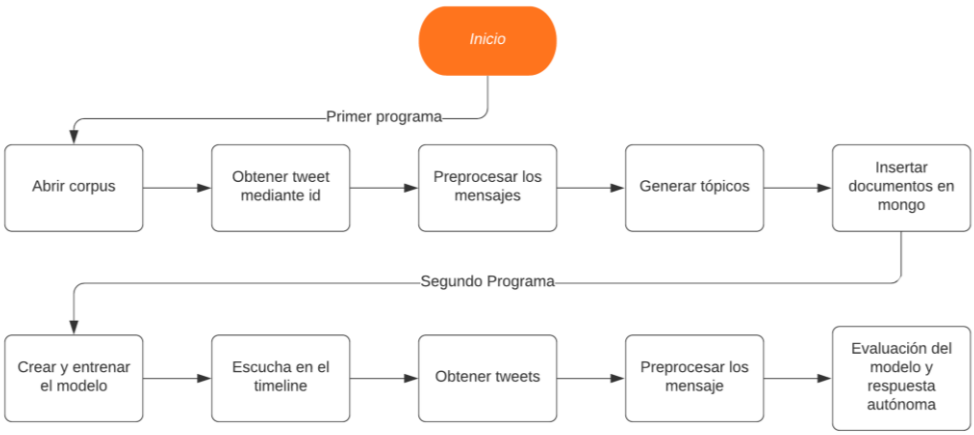

Figura 7: Diagrama de flujo

Para facilitar el desarrollo y asegurarnos de cubrir todos los puntos, el proyecto consta de tres subprocesos: la integración de la API Tweepy, el entrenamiento del modelo de Gensim y la creación de la base de datos de MongoDB.

4.2 Tweepy

El primer paso a realizar para este proyecto fue la creación de un programa para integrar la api de Tweepy. Para poder completar este punto se realizaron varias tareas. En primer lugar, se creó un collab en Google, lo que nos permite ejecutar código en la nube sin necesidad de estar descargando librerías, una herramienta bastante útil cuando se está comenzando a probar las tecnologías al desarrollar un programa.

Antes de crear el código del programa es necesario realizar un paso previo para que la api pueda acceder a nuestra cuenta de twitter y trabajar con la información interna, la creación de una cuenta de desarrollador. Accedemos a la página twitter para desarrolladores: https://developer.twitter.com/en y procedemos a crear una cuenta. Hay que aclarar que no cualquier persona puede crear una cuenta, y en función del tipo de cuenta se te conceden una serie de permisos u otros. En nuestro caso tuvimos que especificar los datos de la cuenta, nuestra situación actual (estudiante), la finalidad de la cuenta de desarrollador, así como crear un proyecto nuevo y contestar a un breve cuestionario con cuestiones relacionadas con la forma con la que trataremos los datos de Twitter, la finalidad de la obtención de estos datos y el uso que recibirá el programa que vamos a crear.

Tras unos días de espera la solicitud fue aceptada y ya podíamos acceder a la cuenta

de desarrollador. El principal uso de esta cuenta es obtener unas credenciales únicas para cada cuenta que permiten a Tweepy acceder a la red social, ya que el acceso es imposible realizarlo mediante el usuario y la contraseña por falta de seguridad.

La interfaz principal de la página de desarrollador es la siguiente:

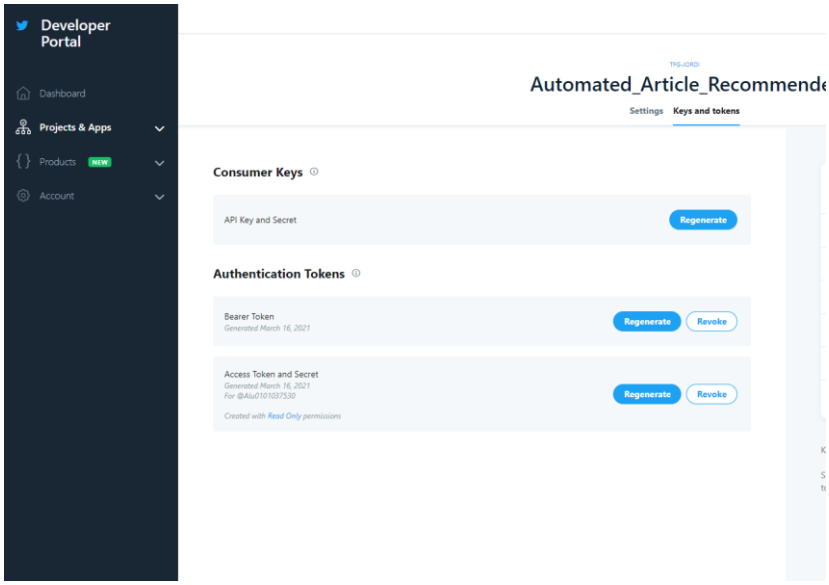

Figura 8: Panel de desarrollador de Twitter

Con nuestro proyecto creado, accedemos a las credenciales que se nos han generado, estas se encuentran en el cuadro de 'Consumer Keys'. Twitter se toma las vulnerabilidades de seguridad con bastante seriedad, por lo que una vez has copiado tus credenciales no podrás volver a verlas, en caso de necesitarlas nuevamente es necesario regenerar unas nuevas.

Ya tenemos los datos para acceder a la cuenta. Creamos cuatro strings que contienen los valores de las 'Consumer Keys' y un atributo que contendrá la autenticación, necesaria para realizar cada operación en la red social.

```
auth = twt.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = twt.API(auth, wait_on_rate_limit=True,wait_on_rate_limit_notify=True)
```

Figura 9: Código para la autenticación de Twitter

Con el acceso funcionando correctamente, el primer ejemplo para comprobar la

obtención de información es el comando ‘api.me()’, que permite obtener los datos de nuestro perfil.

Con la comprobación de que los datos obtenidos son correctos, pasamos a obtener el estado de un tweet y responder al mismo, para ello necesitamos un corpus con ejemplos de estados. El corpus seleccionado consta de un documento con tres columnas, una para el id del tweet, otro con el nombre del author y una última con un hash.

```
741026538825650176,agarzon,7a31bd5b510bafb8a24597b4ad9c346d
741146409899065344,agarzon,6ed053dce9f43c19e7af2ac0f1b05755
741146313933361152,agarzon,9993640d58919ba3607c6b51b4a3f1fc
741147335628083200,agarzon,ba42e7c4be0e505ae456c515b9ba3825
741163656382152704,agarzon,d08d154a06dd13ed5966299293c73436
741180537604345856,agarzon,7ef360c07bd079f47cc30a08c01abc7d
741241254034460672,agarzon,08c4cfe32e0a309d5be299d6d464d3eb
741241088447533056,agarzon,170528123d310f83774b0c20db2defb6
741241068528799744,agarzon,0e7d94fe39ba109c5320819388af25b7
741241010328571904,agarzon,5ffc932584595b9253594c583cba3c72
741246205087387648,agarzon,eabc5d1dca96324ab20b8a6b15d90843
741315385794912260,agarzon,0a777ac3c4c9a8a0cfccd060334a108e
741319928935022592,agarzon,2be7ca9f7d1b06ab1fdff18cdb39dd42
741344802449575936,agarzon,ed876ffb6d140e22d3142decc09fcd90
741367482653528064,agarzon,f1892cf35436e1edc6b5a23daf2141c5
```

Figura 10: Corpus sobre tweets de políticos

Para poder acceder a la columna que necesitamos, creamos una función para abrir el archivo y leer estos valores. Esto lo conseguimos mediante la función ‘smart_open’ que nos aporta python. Abrimos el fichero en modo lectura, especificando la coma como delimitador de las columnas, y por cada fila obtenemos el contenido del mensaje. Para ello empleamos la función ‘get_status(id)’, que nos devuelve un documento con todos los datos relacionador con el contenido del tweet y su autor.

```
with open('/content/tweets_delimitados.csv', 'r') as csv_file:
  csv_reader = csv.reader(csv_file, delimiter=',')
  for row in csv_reader:
    tweet_content = api.get_status(row[0])
    print (tweet_content.text)
```

Figura 11: Código que obtiene tweets mediante su id

Con esto ya tendríamos suficiente para el programa que se encargaría de obtener el contenido del tweet del corpus, pero aún nos queda crear una función para la lectura y respuesta de mensajes.

Creamos una clase que lee en directo nuestro timeline. Aprovechando las funciones creadas anteriormente, las claves de acceso, y algún dato más obtenido de la documentación de Tweepy se ha podido crear un stream de escucha en Twitter.

```
class TweetsListener(twt.StreamListener):

  def on_connect(self):
    print ("Estoy conectado")

  def on_status(self, status):
    if hasattr(status, "retweeted_status"):  # Check if Retweet
        try:
            print("Tweet: \n" + "Nombre: \t" + status.user.name + "\nContent: \t" + status.retweeted_status.extended_tweet["full_text"] + "\n\n")
        except AttributeError:
            print("Tweet: \n" + "Nombre: \t" + status.user.name + "\nContent: \t" + status.retweeted_status.text + "\n\n")
    else:
        try:
            print("Tweet: \n" + "Nombre: \t" + status.user.name + "\nContent: \t" + status.extended_tweet["full_text"] + "\n\n")
        except AttributeError:
            print("Tweet: \n" + "Nombre: \t" + status.user.name + "\nContent: \t" + status.text + "\n\n")

  def on_error(self,status_code):
    print("Error" ,status_code)

consumer_key="9quvZFkRgh8sqxJoeCXTaZgpU"
consumer_secret="Zac5d1DRrRmdroP4uHnHI9GPdMos0kiAMR2iia4wAbXjO6cnX6"
access_token="1371888320729509888-3P88MYMwT8mzL6alSVGEZq0yBOfFCb"
access_token_secret="wElqf2TElSSANwDdx1wK8UYUpFNQ9rC12B78uSWR0ZopK"

auth = twt.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = twt.API(auth, wait_on_rate_limit=True,wait_on_rate_limit_notify=True)

stream = TweetsListener()

streamingApi = twt.Stream(auth=api.auth, listener=stream)
```

Figura 12: Código para escuchar en directo en Twitter

4.3 Gensim

Para integrar Gensim, como usaremos un fichero para leer los contenidos de los mensajes que queremos entrenar, reciclamos la función de lectura del punto anterior cambiando algunos elementos.

Gensim viene con una función de preprocesado para los textos, pero presenta cierta carencia de vocabulario cuando se usa el idioma español, por ello se ha creado una función alternativa que emplea la librería NLTK. Esta función realiza una serie de pasos con cada texto procedente del fichero, que se resumen en:

- En caso de que el tweet contenga un enlace a algún tipo de archivo, este se suprime ya que únicamente aportaría ruido al modelo.

- Separamos el texto en un token por palabra y lo pasamos a minúsculas.
- Quitamos aquellos tokens que correspondan a signos de puntuación o palabras que carecen de significado en el español.
- Lematizamos el texto restante para obtener un resultado más homogéneo.

Con la función creada, la sustituimos donde corresponda para utilizarla en lugar de la original de Gensim.

```
def complete_preprocess(text):
    #Definimos las stop words
    sw = stopwords.words('spanish')

    #Eliminamos en caso de que haya enlaces añadidos
    without_link = text.split("https://")[0] #El [0] lo que hace es quedarse con lo anterior a la expresión

    #Tokenizamos el texto
    with_tokenize = word_tokenize(without_link)

    #Ponemos en minuscula
    with_lower = [w.lower() for w in with_tokenize]

    #Quitamos los signos de puntuacion (not in punctuation) y las stopwords (not in sw)
    without_punt_sw = [w for w in with_lower if w not in punctuation and w not in sw]

    #Lematizamos el texto restante
    wnl = WordNetLemmatizer()
    with_lmt = [wnl.lemmatize(w) for w in without_punt_sw]

    return with_lmt
  # Fin de la función
```

Figura 13: Función de preprocesado de mensajes

Creamos una variable que almacenará los textos procesados del corpus correspondiente.

```
def read_corpus(fname, tokens_only=False):
    with smart_open.open(fname, encoding="iso-8859-1") as f:
        for i, line in enumerate(f):
            tokens = gensim.utils.simple_preprocess(line)
            if tokens_only:
                yield tokens
            else:
                # For training data, add tags
                yield gensim.models.doc2vec.TaggedDocument(tokens, [i])

train_corpus = list(read_corpus(lee_train_file))
```

Figura 14: Función de Gensim para leer y tokenizar corpus

El siguiente paso es crear el modelo que usaremos para entrenar los textos. Está

función requiere el tipo que se va a usar, en nuestro caso es '*Doc2Vec*', además de otros tres atributos:

- Vector_size: define el tamaño que tendrá el vector que usaremos para inferir los textos de la prueba.
- Min_count: este valor contempla el número mínimo de veces que debe aparecer un token para que se use en el aprendizaje, esto ayuda a descartar palabras innecesarias o carentes de significado en el contexto del modelo.
- Epochs: Es el número de veces que se repetirá el entrenamiento. Cuanto mayor sea el valor más pruebas hará el modelo.

```
model = gensim.models.doc2vec.Doc2Vec(vector_size=50, min_count=2, epochs=40)
```

Figura 15: Modelo de doc2vec de Gensim con sus parámetros de creación

El modelo nos permite obtener un vocabulario con términos presentes en el corpus. Para generarlo usamos la función '*build_vocab(train_corpus)*'.

Entrenamos el modelo con la función 'train', el cual empleará los valores que se han especificado a la hora de crearlo. Tras haber realizado el entrenamiento el número de repeticiones que hemos declarado, ya podemos comprobar los resultados con textos que no se han empleado para el corpus de entrenamiento.

Se crea una variable que almacena un vector que codifica al tweet al que se pretende dar respuesta de forma automática. Este proceso se lleva a cabo a través de la función 'infer_vector', pasándole el texto que queramos probar como parámetro. El problema se convierte en el cálculo de similitudes entre los vectores generados por el modelo Doc2Vec y los vectores inferidos.

Emplearemos la función 'most_similar' para comparar los textos a través de su representación vectorial. Esta función compara el vector que hemos inferido del texto de prueba anterior con cada uno de los vectores de corpus de entrenamiento, generando como resultado una lista ordenada de más similar a menos con cada texto que compone el conjunto, esto nos permite obtener un resultado más amplio para aquellos casos que deseemos más de una posible solución.

```
doc_id = random.randint(0, len(test_corpus) - 1)
inferred_vector = model.infer_vector(test_corpus[doc_id])
sims = model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))

print('Test Document ({}): «{}»\n'.format(doc_id, ' '.join(test_corpus[doc_id])))
print(u'SIMILAR/DISSIMILAR DOCS PER MODEL %s:\n' % model)
for label, index in [('MOST', 0), ('MEDIAN', len(sims)//2), ('LEAST', len(sims) - 1)]:
    print(u'%s %s: «%s»\n' % (label, sims[index], ' '.join(train_corpus[sims[index][0]].words)))
```

Figura 16: Código para obtener documentos similares

Como en este proyecto únicamente necesitaremos el texto más similar, accederemos a la posición inicial de la matriz resultante de la función.

Con esto ya tenemos lo suficiente para obtener una respuesta a los tweets que obtengamos del timeline de nuestra cuenta de twitter.

4.4 Base de datos MongoDB

Como se comentó en anteriores puntos, usaremos la base de datos no relacional de Mongodb ya que nos da la posibilidad de trabajar con colecciones de documentos.

Una vez hemos añadido las dependencias ya podemos empezar a usar la base de datos.

Inicialmente se optó por utilizar una base de datos localizada en una máquina virtual, pero debido a una serie de circunstancias se decidió cambiar a una base de datos en la nube, ya que es más fácil de acceder, y puede ser compartida fácilmente sin necesidad de gestionar usuarios. Para ello se empleó el cloud de Mongodb, una plataforma online que ofrece la posibilidad de crear bases de datos propias, y además vienen optimizadas para su acceso en varios lenguajes de programación.

Por tanto, se creó una cuenta en esta plataforma y se procedió a configurarla para poder almacenar los datos necesarios. A la hora de crearla, se muestran una serie de pasos para seleccionar un lenguaje de programación, y como resultado te devuelve una función en dicho lenguaje con un enlace para acceder a esta base.

```
client = pymongo.MongoClient("mongodb+srv://jordi:
<password>@cluster0.ntm40.mongodb.net/myFirstDatabase?retryWrites=true&w=majority")
db = client.test
```

Figura 17: Instrucciones para autenticarse en MongoDB

El primer paso es crear una nueva base de datos en la que almacenaremos la colección de documentos correspondiente. En el lenguaje de Python es tan sencillo como escribir ‘client.database’, donde el valor ‘database’ es el nombre que recibirá la base de datos. Este comando accede a dicha base, o la crea en caso de no encontrarla.

```
db = client.twitter
```

Figura 18: Comando para acceder a la base de datos

La siguiente tarea es crear la colección donde se introducirán los documentos, usamos el mismo comando mencionado en el párrafo anterior, pero en este caso el valor será el nombre de la colección. De igual manera que en el paso previo, el resultado se debe almacenar en una variable, de esta manera nos ahorramos tener que escribir el comando cada vez que queramos acceder a la colección creada.

Para los documentos que se usarán en este proyecto se han definido al menos tres campos necesarios:

- tweet_id: Es un Int64. Este campo almacena el número que identifica el tweet que estamos insertando, para poder acceder al mensaje en caso de que sea necesario.
- tweet _user: Campo de tipo String. Almacena el nombre de usuario del autor del mensaje.
- tweet _text: Campo de tipo String. Este atributo contiene el texto del tweet que ha sido procesado.

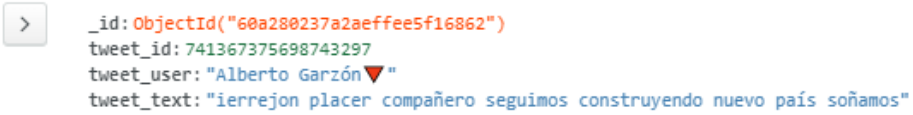

Figura 19: Contenido de un documento de la base de datos

Para poder insertar un nuevo documento en la colección se debe insertar con un atributo denominado diccionario. Este elemento se define entre corchetes y contiene una clave con el nombre del campo que se va a añadir y su valor correspondiente, ambos se escriben entre comillas simples.

Con el diccionario definido, lo único que hay que hacer es llamar a la función

‘*insert_one*’ y pasarle el elemento como parámetro.

```
# Escribir en la base de datos
mydict = {'tweet_id': tweet_content.id,
          'tweet_user': tweet_content.user.name,
          'tweet_text': content}
recommendations.insert_one(mydict)
```

Figura 20: Código para insertar un nuevo documento

Repetimos esta acción con cada tweet existente en el corpus, a mayor número de entradas en la colección más eficiente será la herramienta.

Con esto ya tendríamos cada una de las partes del proyecto operativas para poder desarrollar el programa.

4.5 Integración y resultado final

Con todas las partes anteriores desarrolladas, llega el momento de unificarlas para que la herramienta haga su trabajo. Como comentamos se crearán dos programas, el primero se encargará de obtener lo tweets que se encuentran en el corpus para procesarlos y almacenarlos en la base de datos, y el segundo se deberá conectar a la red social de Twitter para realizar la escucha activa y de esta manera poder relacionar los tweets entrantes con la base de datos y ofrecer una respuesta.

Esta decisión de dividir el proyecto en dos programas se debe a que la primera parte, relacionada con el rellenado de la base de datos, solo se realiza una vez, y debido al coste de realizar conexiones a la nube de Mongodb, cuantas menos se realicen más rápido será el programa de escucha. De igual manera, en el segundo programa se realizarán conexiones, pero estas se reducirán a las mínimas indispensables, además, se realizarán en modo de lectura, lo cual es menos costoso.

4.5.1 Programa de ingesta de tweets

Para el correcto funcionamiento de las herramientas añadimos las funciones de conexión a Twitter y a MongoDB.

Con este paso previo ya podemos pasar a procesar los textos, para lo cual emplearemos la función de procesado que se ha creado. La copiamos en el programa junto con la función ‘smart_open’ para abrir el archivo. En este programa

modificaremos la última función para que cree además un nuevo archivo en modo de escritura en el cual almacenaremos los textos procesados, de esta forma no será necesarios acceder a la base de datos innecesariamente para entrenar el modelo en el segundo programa, ahorrándonos tiempo.

El funcionamiento de esta sección del programa será la siguiente:

- Se abre el archivo de lectura que contiene los id de los tweets.
- Sin cerrarlo, se abre el nuevo archivo en modo de escritura.
- Por cada línea existente en el corpus, se obtiene el campo correspondiente a la primera columna, que contiene el id del mensaje.
- Se llama a la función '*get_status*' de tweepy para obtener la información.
- Se procesa el contenido del mensaje y se guarda en una nueva variable.
- Con el texto procesado, se escribe una nueva línea en el fichero de escritura.
- Se crea un diccionario con los campos a rellenar en el documento de la colección de mongo a partir de la información obtenida del tweet.
- Se inserta dicho diccionario en la base de datos en la nube.

```
with open('corpus_reducido.txt', 'r') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

# Abrimos el archivo donde guardaremos el contenido de los tweet para entrenar el modelo.
    with open('new.txt', 'a') as new_file:  # 'a' para hacer append del texto y no sobreescribir
        # Introducimos el contenido de cada tweet obtenido en el archivo.
        for row in csv_reader:
            i += 1
            try:
                tweet_content = api.get_status(row[0])
            except twt.TweepError as e:
                print(e, 'on tweet numher: ', i)
                continue

            # Escribir en el fichero
            preprocess_text = complete_preprocess(tweet_content.text)
            new_file.write(" ".join(preprocess_text))
            new_file.write("\n")
            content = " ".join(preprocess_text)

            # Escribir en la base de datos
            mydict = {'tweet_id': tweet_content.id,
                      'tweet_user': tweet_content.user.name,
                      'tweet_text': content}
            recommendations.insert_one(mydict)
```

Figura 21: Función encargada de obtener el tweet, procesarlo e insertarlo en la base de datos

Una vez se han procesado e insertado todos los mensajes, los ficheros se cierran y el programa finaliza.

4.5.2 Programa de respuesta autónoma

Para este programa se debe incluir todo lo relacionado con el modelo de entrenamiento:

- La función de lectura encargada de procesar y tokenizar los textos. Se selecciona el fichero de escritura creado en el punto anterior para el aprendizaje.
- Se crea el modelo con los parámetros que se deseen, se genera el vocabulario, y se entrena.

Además, debemos incluir la función de procesado que hemos creado, ya que los tweets que obtengamos en directo del timeline de Twitter probablemente contengan bastante ruido.

Con estas tareas funcionando es momento de añadir la clase responsable de la escucha, la cual contiene las siguientes funciones:

- On_connect: Comprueba que la api se ha conectado correctamente a la cuenta.
- On_status: Esta es la función principal, se encarga de obtener el mensaje del timeline, procesarlo, obtener el documento más similar y contestar al mensaje.
- On_error: Mostrará un mensaje con un código de erro en caso de que alguna de las funciones anteriores falle.

Para mejorar la comprensión sobre el funcionamiento del programa, se procede a definir el contenido del '*On_status*':

- En primer lugar, se obtiene el contenido del mensaje. Dado que en Twitter existe la posibilidad de hacer retweet incluyendo un mensaje, es necesario comprobar qué tipo de mensaje estamos recibiendo, para saber cómo buscar el campo del contenido.
- Con el texto obtenido llamamos a la función de procesado para tokenizar y

eliminar ruido.

- Se crea un vector con los tokens y, mediante el método '*most_similar*', obtenemos el documento más similar de nuestra base.

Con esto ya tendríamos operativo el segundo programa, el cual genera respuestas como la siguiente.

Figura 22: Respuesta a un tweet

4.5.3 Análisis de resultados

Para analizar la fiabilidad del programa implementado, empleamos un corpus de datos de los cuales, aproximadamente 22.000 conforman la muestra de entrenamiento insertada en la base de datos, y un conjunto aleatorio de 60 mensajes han sido seleccionados del resto del corpus para comprobar la exactitud del modelo. Para tener un campo con el que comparar los resultados, se ha usado un pequeño script que genera 2 tópicos con los que clasificar los documentos de la base. El motivo de no incrementar el número de tópicos se debe a que, pese al gran número de documentos, las palabras obtenidas para conformar los tópicos no son demasiadas, y si se aumentara generaría un tópico por palabra, perdiendo el sentido de esta funcionalidad.

Para el estudio se comprueban cuantos mensajes de la prueba coinciden en tema con el tópico, y un valor entre 0 y 1 sobre la similitud entre el mensaje y su correspondiente.

- Con un total de 20 entrenamientos:

    - Aciertos en el tópico: ~64%
    - Porcentaje medio de similitud: ~0.75
- Con un total de 40 entrenamientos:
    - Aciertos en el tópico: ~80%
    - Porcentaje medio de similitud: ~0.76

Se puede observar que un aumento en el número de entrenamientos no varía en el porcentaje de similitud, esto se podría deber a que, con la cantidad de documentos de la base de datos, encuentre coincidencia sin demasiada dificultad, sin embargo, si se aprecia un aumento en los aciertos sobre el tópico del que trata. Esto es un punto a favor para el modelo más entrenado, ya que únicamente genera ventajas, pero ha sido necesario duplicar el número de entrenamientos para aumentar en un 16% el acierto de tópicos, por lo que un excesivo aumento en los entrenos no generaría un cambio drástico, e incluso sería posible que generara resultados inferiores.

```
Documento de prueba:  vamos derogar reforma laboral universalizar sanidad sanchezcastejon unsíporelcambio votapsoe
Documento similar:  recuperar universalidad sanidad pública invertir 7 pib sanidad sanchezcastejon debate13j
Similaridad:  (7312, 0.7593206763267517)
Tópico:  PedroSanchezPsoe
```

Figura 23: Ejemplo de obtención de un documento y su porcentaje de similitud.

Capítulo 5
Problemas encontrados

5.1 Limitaciones de tweepy

Uno de los primeros problemas surgió a la hora de testear el programa encargado de procesar los textos que se insertarían en la base de datos. Este proceso es bastante largo, ya que consta de decenas de miles de tweets, y durante la ejecución del programa, tras realizar un cierto número de peticiones el programa se cancelaba sin motivo aparente.

En un principio se desconocía cuál era el origen del problema, ya que el programa no contaba con una función para mostrar los errores. Tras crear una, se obtuvo un código de error cuando sucedía esto. Buscando en la documentación de desarrolladores de twitter se observó que existe una serie de restricciones en cuanto a número de peticiones se refiere, establecida para limitar la obtención de información.

En nuestro caso nos afectaba en la obtención de tweets, el cual tenía un límite de 900 tweets por usuario, tras el cual se debería esperar un periodo de 15 minutos para poder realizar peticiones de nuevo.

Para solucionar este conflicto, la api de Tweepy propone una solución en su documentación, que consta de añadir un parámetro denominado ‘wait_on_rate_limit’ a la hora de crear la Api. Esto consigue que el programa respete el intervalo de tiempo durmiendo el programa y evitando así que se termine el proceso.

5.2 Conexión con MongoDB

Como se comentó al comienzo del desarrollo, se seleccionó el entorno de desarrollo en la nube de Google para programar la aplicación mediante un cuaderno de Jupyter.

Durante la gran parte de codificación de los procesos no se produjeron complicaciones, pero a la hora de realizar la conexión a la base de datos surgió una dificultad. Dado que la base de datos no se encontraba en la máquina local, la manera de conectarse requería una librería extra.

El problema surge porque este entorno de desarrollo no contiene esta librería, y no es posible instalarla ya que los archivos se eliminan cuando se reinicia la sesión.

La primera idea fue eliminar este campo de la dirección de conexión y hacerlo de forma más básica, pero esto no permitía obtener información de la base de datos, el programa se quedaba buscando hasta que saltaba un error por superar el tiempo máximo de espera.

La solución por tanto se basa en migrar el programa a una máquina virtual en el cual podamos instalar las librerías localmente para evitarnos este tipo de problemas.

5.3 Contenido de los tweets

El principal problema en relación con la fiabilidad de la similitud de documentos se produce por el contenido de los tweets. La jerga que se emplea en esta red social genera mucho ruido en el análisis, y el procesado carece de funciones para eliminar ciertos elementos. Esto sucede por ejemplo con algunos emoticonos que tras el procesado seguían presentes, al igual que con algunos enlaces a sitios externos, palabras como ‘rt’ o ‘fav’. Además, varios mensajes estaban escritos en catalán, lo que reducía las capacidades del modelo y generaba carencia al pasar el procesado, el cual apenas modificaba el texto.

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

La idea inicial acerca de la finalidad que tendría este proyecto se ha cumplido, la herramienta es capaz de responder sin necesidad de ayuda humana más allá de iniciar el programa.

Durante el desarrollo del programa se han ido descubriendo diversas herramientas, librerías y demás elementos que han servido de mejore, aunque no estuvieran contempladas en un principio.

Con respecto a las distintas partes en las que se ha dividido el proyecto, estas se han cumplido dentro del tiempo general establecido. Aunque cabe destacar, como apunte personal, que en ocasiones ha resultado de mayor dificultad las partes que en teoría deberían haber sido más fáciles de implementar, esto se ha debido principalmente a causa de emplear un entorno de programación que no es compatible con el tipo de proyecto, o a que la documentación de una librería no se encontraba actualizada, como sucedió con la implementación del modelo de aprendizaje Doc2Vec de gensim.

En cuanto a mi visión personal sobre lo que se ha tratado en este proyecto, la inteligencia artificial es un ámbito que me ha llamado la atención, pero no he tenido oportunidad de estudiarlo mucho en la carrera, por lo que me intrigó hacerlo desde que leí la descripción del trabajo. Aunque esta herramienta no involucra muchos conceptos de entre los tipos de aplicaciones que hay para la IA , me ha permitido aprender bastantes cosas durante el periodo que tuve de documentarme. Además, Python es un lenguaje que me gusta bastante por la sencillez, pero a la vez potencia, que tiene a la hora de programar.

6.2 Líneas futuras

Aunque el programa cumple la función establecida, existen varios puntos para tener

en cuenta si se quisiera mejorar la herramienta.

En primer lugar, si se quisiera seguir empleando el español como idioma para la herramienta, sería conveniente buscar o crear un vocabulario más extenso en referencia a las stopwords. Librerías como NLTK o las funciones de preprocesado de gensim son bastante completas cuando se emplean en su lenguaje nativo, pero carecen de numerosos términos cuando se trata en español. En mi opinión sería conveniente crear además un pequeño vocabulario relacionado con la terminología que se emplea en cada red social, ya que en muchas ocasiones no eliminaban palabras que como humanos sabemos que no aportan información más que para usarla como jerga dentro de la red social.

En cuanto a capacidades de la api, Twitter permite hacer numerosas acciones más allá de contestar un tweet. Se podría modificar la aplicación para que envíe mensajes directos a usuarios que hablen sobre un tema que sea de interés para la entidad que use la aplicación.

Otra utilidad que se le podría dar para empresas que presten servicios, por ejemplo, sería analizar el contenido de los usuarios para deducir a cuáles de ellos podrían serle de utilidad nuestros servicios, y en caso de obtener un resultado positivo, establecer que el programa comience a seguir a esa persona, creando de esta forma un público objetivo para nuestra empresa a través de las redes sociales.

Capítulo 7
Summary and Conclusions

7.1 Conclusions

The initial idea about the purpose that this project would have has been fulfilled, the tool can respond without the need for human help beyond starting the program.

During the development of the program, various tools, libraries, and other elements have been discovered that have served as improvements, although they were not contemplated at first.

Regarding the different parts in which the project has been divided, these have been fulfilled within the established general time. Although it should be noted, as a personal note, that sometimes the parts that in theory should have been easier to implement have been more difficult, this has been mainly due to using a programming environment that is not compatible with the type of project, or that the documentation of a library was not up to date, as happened with the implementation of the Doc2Vec learning model from Gensim.

As for my personal vision of what has been covered in this project, artificial intelligence is an area that has caught my attention, but I have not had the opportunity to study it a lot in my career, so I was intrigued by doing it since I read the project description. Although this tool does not involve many concepts from among the types of applications that there are for AI, it has allowed me to learn quite a few things during the period I had to document myself. In addition, Python is a language that I quite like for its simplicity, but at the same time power, that it has when programming.

7.2 Future lines

Although the program fulfills the established function, there are several points to consider if you want to improve the tool.

In the first place, if you want to continue using Spanish as the language for the tool,

it would be convenient to search for or create a more extensive vocabulary in reference to stop words. Libraries like nltk or gensim's preprocessing functions are quite complete when used in their native language, but they lack numerous terms when it comes to Spanish. In my opinion, it would also be convenient to create a small vocabulary related to the terminology that is used in each social network, since on many occasions they did not eliminate words that as humans we know that do not provide information other than to use it as jargon within the social network.

Regarding api capabilities, Twitter allows you to do numerous actions beyond replying to a tweet. The application could be modified to send direct messages to users who talk about a topic that is of interest to the entity using the application.

Another utility that could be given to companies that provide services, for example, would be to analyze the content of users to deduce which of them our services could be useful to, and in case of obtaining a positive result, establish that the program starts to follow that person, thus creating a target audience for our company through social networks.

Capítulo 8
Presupuesto

Para generar el presupuesto de este proyecto nos centraremos en las horas de trabajo invertidas en él, con su correspondiente desglose. Dado que las herramientas utilizadas son de carácter gratuito, el único material cuyo coste se incluirá en el presupuesto es:

- Equipo: Ordenador portátil o de sobremesa, a elección del cliente. Características recomendadas: Procesador quad core, 8 gb de memoria ram, gráfica de video con virtualización de gpu para soportar la máquina virtual.

El coste total del proyecto es de 3500€, desglosado a continuación:

Tarea	Horas	Precio/Hora	Total
Documentación y lectura bibliográfica.	25	20€	500
Planificación de desarrollo	15	20€	300
Integración Tweepy	20	20€	400
Desarrollo con Gensim (modelo y tópicos)	40	20€	800
Implementación MongoDB	10	20€	200
Insertar documentos en la base de datos	10	20€	200
Test y evaluación de resultados	25	20€	500
Equipo			600€

Capítulo 9
Bibliografía

[1] M. A. Alonso and D. Vilares, "A review on political analysis and social media," Proces. leng. nat., vol. 56, no. 0, pp. 13–24, 2016.

[2] C. N. H. Miranda, J. A. G. Luna, and D. Salcedo, "Minería de Opiniones basado en la adaptación al español de ANEW sobre opiniones acerca de hoteles," Proces. leng. nat., vol. 56, no. 0, pp. 25–32, 2016.

[3] N. Padrón Díaz, "Automatización de respuestas lógicas a tweets mediante PLN," Universidad de La Laguna, 2020.

[4] Twitter Help Center, "Información sobre las API de Twitter," *Twitter.com*, 03-Mar-2020. [Online]. Available: https://help.twitter.com/es/rules-and-policies/twitter-api. [Accessed: 27-Mar-2021].

[5] "Gensim: topic modelling for humans," *Radimrehurek.com*. [Online]. Available: https://radimrehurek.com/gensim/. [Accessed: 27-Mar-2021].

[6] "Procesamiento del lenguaje natural ¿qué es? - IIC," *Uam.es*, 17-Oct-2017. [Online]. Available: https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/. [Accessed: 27-Mar-2021].

[7] S. J. Rodríguez Martín, "Extracción de información a partir de correos electrónicos usando técnicas de procesamiento del lenguaje natural," 2017.

[8] "¿Qué es el Procesamiento de Lenguaje Natural y cómo aplicarlo?," 02-Nov-2018. [Online]. Available: https://www.youtube.com/watch?v=5c0qIh54uqE. [Accessed: 27-Mar-2021].

[9] A. Naveira, "Historia de Twitter: de un comienzo brillante a los rumores sobre su futuro incierto," *Marketing4ecommerce.net*, 23-Jan-2020. [Online]. Available: https://marketing4ecommerce.net/historia-de-twitter/. [Accessed: 10-Apr-2021].

[10] N. D., "¿Qué son los bots o seguidores fantasma que asedian el Twitter de los políticos?," *La Voz de Galicia*, 09-Sep-2014. [Online]. Available: https://www.lavozdegalicia.es/noticia/tecnologia/2014/09/09/bots-seguidores-fantasmas-asedian-twitter-politicos/00031410261277161827597.htm. [Accessed: 10-Apr-2021].

[11] "TÉRMINOS y PALABRAS Frecuentes en TWITTER," *Ignaciosantiago.com*, 12-Jul-2013. [Online]. Available: https://ignaciosantiago.com/terminos-twitter/. [Accessed: 10-Apr-2021].

[12] IMF Business School, "El procesamiento del lenguaje natural: sus modelos y sus usos prácticos," *Imf-formacion.com*, 08-Jun-2020. [Online]. Available: https://blogs.imf-formacion.com/blog/tecnologia/procesamiento-lenguaje-natural-modelos-usos-practicos-202006/. [Accessed: 10-Apr-2021].

[13] G. Romeo, "Bases de datos NoSQL," *Seas.es*, 04-Nov-2013. [Online]. Available: https://www.seas.es/blog/informatica/bases-de-datos-nosql/. [Accessed: 10-Apr-

2021].

[14] “NoSQL la evolución de las bases de datos,” *Com.mx*. [Online]. Available: https://sg.com.mx/revista/28/nosql-evolucion-bases-datos. [Accessed: 10-Apr-2021].

[15] “Tweepy,” *Tweepy.org*. [Online]. Available: https://www.tweepy.org/. [Accessed: 11-Apr-2021].

[16] J. A. Mora, “Primeros pasos con Tweepy,” *Wordpress.com*, 20-Jun-2019. [Online]. Available: https://jantoniomora.wordpress.com/2019/06/20/primeros-pasos-con-tweepy/. [Accessed: 11-Apr-2021].

[17] S. Prabhakaran, “Gensim tutorial - A complete beginners guide - ML+,” *Machinelearningplus.com*, 16-Oct-2018. [Online]. Available: https://www.machinelearningplus.com/nlp/gensim-tutorial/. [Accessed: 11-Apr-2021].

[18] Datademia, “¿Qué es MongoDB?,” *Datademia.es*, 02-Dec-2020. [Online]. Available: https://datademia.es/blog/que-es-mongodb. [Accessed: 24-Apr-2021].

[19] “What Is MongoDB?,” *Mongodb.com*. [Online]. Available: https://www.mongodb.com/what-is-mongodb. [Accessed: 24-Apr-2021].

[20] N. Colectiva, “Tipos de Datos que podemos usar en JSON,” *Nubecolectiva.com*. [Online]. Available: https://blog.nubecolectiva.com/tipos-de-datos-que-podemos-usar-en-json/. [Accessed: 24-Apr-2021].

[21] L. A. U. Fernández, “Reducir el número de palabras de un texto: lematización y radicalización (stemming) con Python,” *qu4nt*, 04-May-2019. [Online]. Available: https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa. [Accessed: 27-Mar-2021].

[22] A. Perez, “Python (Un poco de historia y algunas de sus características básicas),” *Datonautas*, 30-Jul-2018. [Online]. Available: https://medium.com/datonautas/python-un-poco-de-historia-y-algunas-de-sus-caracter%C3%ADsticas-b%C3%A1sicas-a685ebcc27db. [Accessed: 24-Apr-2021].

[23] Wikipedia contributors, “Historia de Python,” *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Historia_de_Python&oldid=133215973. [Accessed: 24-Apr-2021].

[24] Wikipedia contributors, “NLTK,” *Wikipedia, The Free Encyclopedia*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=NLTK&oldid=127418331. [Accessed: 24-Apr-2021].