**News App**

**Universidad Complutense de Madrid**

**Facultad de Informática**



Enrique Maestro Mañanes

Manuel Guerrero Moñus

[**Introducción**](#_7mfjcm1nm77) **3**

[**Diseño**](#_h8b2rd67kwrw) **3**

[Agentes](#_spzpkuphobx3) 3

[ChatBot](#_ozf11erz782y) 3

[Analizador](#_zbd3bs9vswg5) 10

[Recomendador](#_ktk4yb86lvha) 13

[Arquitectura y diagramas](#_qn3ufmjkvq8q) 14

[Casos de uso](#_g0cf8e96qadl) 14

[Diagrama arquitectónico](#_ek15x7l3u3s0) 17

[Diagramas de secuencia](#_uvh64y9r7wb) 19

[**Implementación**](#_1lmek874l6mr) **22**

[Spade](#_gzyo4qc20jf6) 22

[ChatterBot](#_39xjz5eio52v) 23

[Scikit-learn](#_rfu5bda4cpsj) 24

[NLTK](#_swtbusfqwlxe) 25

[GoogleNews](#_oro2vcmytfb2) 25

[**Planificación**](#_cmaxsuumsjro) **26**

[**Trabajo futuro**](#_xxipxspokeez) **26**

[**Conclusiones**](#_23b5hgrt91ed) **27**

[**Bibliografía**](#_g6570t3iq3as) **27**

[**Anexo 1: Manual de instalación**](#_16wb01dhqd4x) **28**

[Introducción](#_joa9lxt1072r) 28

[Instalación](#_u004qag8y4y) 28

[Ejecución](#_vae1u1jpn69f) 28

[Desinstalación](#_elt6k11qnx0s) 28

[Ficheros de código fuente](#_8c8e28v3l0db) 29

[Ficheros de recursos](#_gegdt8d0eu67) 30

[**Anexo 2: Manual de usuario**](#_zgnehkp7yj9u) **32**

[Ventana principal](#_f6nu064pz17l) 32

[Ayuda](#_pxxavpqemtm8) 33

[Clasificador](#_8wovvlkpih88) 34

[Analizador](#_mw468rxs4l7p) 35

[Recomendador](#_eoztddr807pd) 36

[Error](#_ylfb0llqdjvb) 37

# Introducción

En este proyecto se ha realizado un ChatBot capaz de clasificar, analizar y recomendar noticias deportivas, utilizando agentes inteligentes como característica principal. Se pretende que a partir de una conversación entre este ChatBot y un usuario, el sistema sea capaz de deducir las necesidades del usuario, devolviendo la información necesaria, es decir, informando al usuario de la categoría en la que se clasifica una noticia, la extracción de la información de una noticia o recomendar noticias sobre un tema concreto.

# Diseño

## Agentes

Para desarrollar este proyecto hemos implementado un sistema con cuatro agentes. Cada agente está implementado para realizar solo una funcionalidad. Los cuatro agentes que utilizamos son los siguientes:

### ChatBot

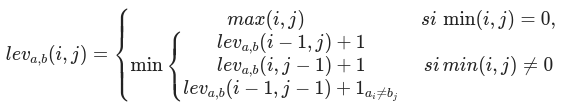
Este agente es el principal del proyecto, se encarga de interactuar con el usuario y analizar el texto que introduce para saber qué es lo que quiere. Cuando reconoce la entrada del usuario como una petición reconocible, entonces realiza alguna de las funcionalidades del proyecto (clasificar, analizar o recomendar), para lo cual pide la información necesaria para realizar dicha funcionalidad y luego manda un mensaje al agente correspondiente para que realice su función, cuando dicho agente le devuelve por mensaje la respuesta, este la reenviará hacia arriba en la arquitectura de software hasta llegar a la vista y serle mostrada al usuario.

El proceso interno que realiza el agente ChatBotAgent al cargarse es el siguiente:

1. Se comprueba si el ChatBot agent ya posee una BDD que albergue las conversaciones del objeto ChatBot, en caso de que exista, este objeto se inicia con ella y no es necesario un período de reentrenamiento.
2. En caso de que el fichero de la BDD con las conversaciones no esté disponible, entonces se creará el fichero automáticamente en la ruta indicada y posteriormente se reentrenará al objeto ChatBot con nuestro propio corpus.

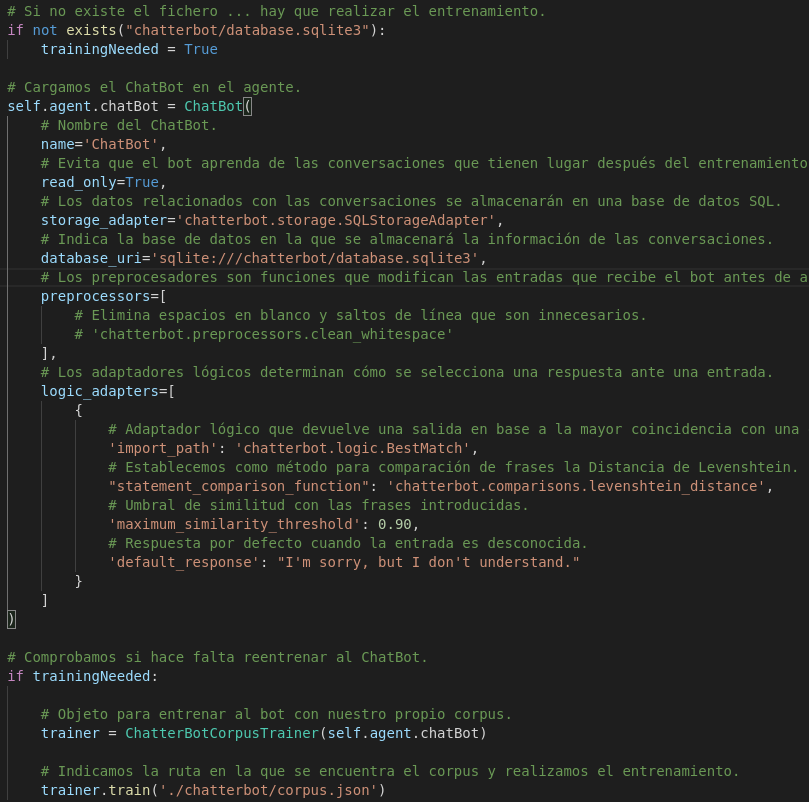
El Chatbot se configura de esta forma:

* read\_only: el bot no aprenderá de las conversaciones post entrenamiento.
* storage\_adapter: las conversaciones del entrenamiento se almacenarán en una BDD de tipo SQL (sqlite3).
* preprocessors: no se ha utilizado ningún preprocesamiento de la entrada de texto.
* logic\_adapter: define el comportamiento del ChatBot ante una entrada.
  + Se indica explícitamente que se ha de buscar la frase almacenada que tenga una mayor coincidencia con la frase introducida por el usuario.
  + Para comparar las frases se utiliza la Distancia de Levenshtein:
    - Método inventado por el científico ruso Vladimir Levenshtein en 1965.
    - Se define como el número mínimo de operaciones (de inserción, eliminación o sustitución) necesarias para transformar una cadena de caracteres en otra.
    - Se calcula siguiendo un algoritmo de programación dinámica:



* Donde LEV A,B (i,j) es la distancia entre los i caracteres de la cadena A y los j de la B.
* La expresión 1Ai=Bj, representa una función que devuelve 0 cuando las dos cadenas de caracteres Ai y Bi​ son iguales, 1 si son diferentes.
* Y la función min representa la realización de alguna de las tres posibles transformaciones, la primera sería la eliminación de un carácter en la cadena A, la segunda la inserción de un carácter y la tercera la realización una sustitución.
  + - El método suele utilizarse en programas que comparan palabras, por ejemplo en correctores ortográficos.
  + Si la similaridad de las frases es del 90% o mayor, entonces se la considera una frase coincidente. El ChatBot siempre devolverá la que tenga un mayor nivel de coincidencia.

El código que representa el proceso de carga y configuración del ChatBotAgent que se ha descrito es:



El código que hace uso de la Distancia de Levenshtein para comparar la entrada del usuario con alguna de las entradas posibles aprendidas durante el entrenamiento es:

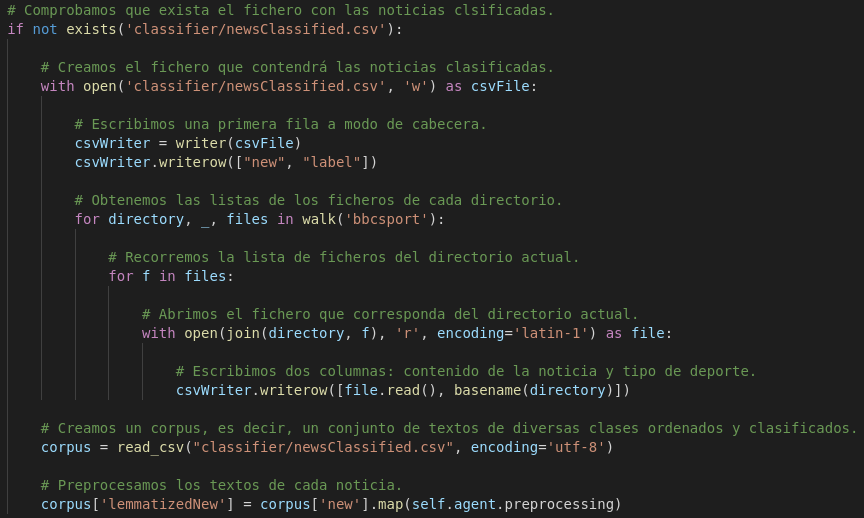


Clasificador

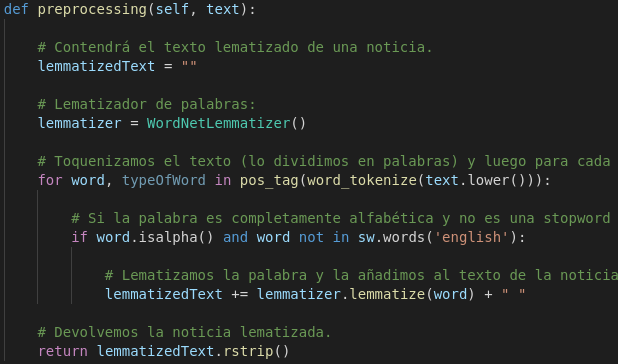
Es el agente que se ocupa de realizar la clasificación de una noticia deportiva en una categoría concreta. Recibe el nombre de la noticia a clasificar a través de los mensajes que proceden del agente ChatBot, una vez recibido, emplea una máquina de vectores de soporte para clasificarla, y después hacerlo, notifica al ChatBot la categoría para que este termine redirigiendo la información hacia la vista y así el usuario pueda ver el resultado de la clasificación.

El proceso de carga del agente realiza las siguientes acciones:

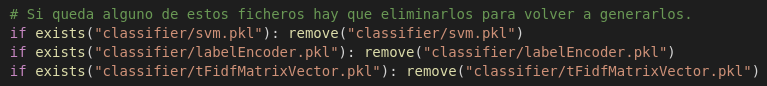
1. Primero se comprueba que exista el fichero “newsClassified.csv”, que contiene dos columnas para cada noticia. En la primera se guarda su texto y en la segunda una etiqueta que indica a qué categoría deportiva pertenece la noticia.
2. Si no existe, entonces se reentrena el modelo:
   1. Se crea el fichero.
   2. Se rellena el fichero realizando un recorrido por el set de noticias deportivas “bbcsport”, este recorrido sirve para rellenar cada fila del fichero csv. El texto de la noticia actual se convierte en el texto de la primera columna de la fila actual del csv y la segunda columna recibe el nombre de la carpeta actual que estamos recorriendo, ya que esta se llama igual que el deporte realizado en la noticia leída.
   3. Una vez que el fichero existe se pasa el texto de todas las noticias por una función de preprocesamiento, esta función crea una nueva columna en el fichero csv que está cargado en memoria.



* 1. La función de preprocesamiento recibe por parámetro el texto de cada noticia del fichero de csv y hace lo siguiente:
     1. Carga un lematizador de palabras.
     2. Tokeniza la noticia a nivel de palabra.
     3. Recorre todas las palabras obtenidas en el proceso de tokenización para comprobar si estas son completamente alfabéticas (A-Z o a-z) y que no sean de tipo stopword. Si esto se cumple, entonces la palabra es lematizada y añadida a una variable de texto que devolverá la función.
     4. El texto de la noticia con las palabras lematizadas es devuelto por la función.



* 1. Como el fichero de csv con los textos se ha generado y no sabemos si se han añadido nuevas noticias o categorías deportivas, por lo que eliminamos la máquina de vector de soporte serializada, también la matriz de vectores TF - IDF de las noticias y el codificador de categorías.



* 1. Llegados a este punto, volvemos a generar la matriz de vectores TF - IDF y el codificador de categorías.
     1. Un objeto TfidfVectorizer genera una matriz (lista de listas) cuyos elementos son vectores cuyos valores representan numéricamente cuán relevante es una palabra para un documento en una colección.
        + TF (Term Frequency): es la frecuencia del término clave en el documento.
          - El TF de un término clave Ki en un documento Dj, representada como TF ij, se define como:

TF ij = 1 + log2(Frequency ij), si Frequency ij>0

TF ij = 0, si Frequency ij=0

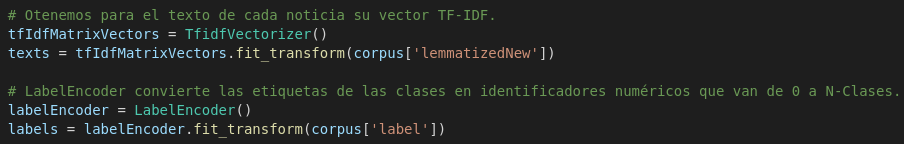
* + - * + La frecuencia de término clave TF ij será proporcional a la frecuencia de Ki en el documento Dj.
      * IDF (Inverse Document Frecuency): es la frecuencia inversa de un término en todos los documentos.
        + La IDF trata de reflejar la importancia de un término clave Ki dentro de una colección, el valor de esta es inversamente proporcional al número de documentos donde aparece el término Ki.
        + Si un término clave aparece en muchos documentos de la colección, entonces este es poco discriminante y por tanto tendrá un peso bajo.
        + Si un término aparece en pocos documentos, entonces es bastante discriminante y tendrá un peso alto.
        + Fórmula → IDF i = log (N / Ni):

N → Nº de documentos en la colección.

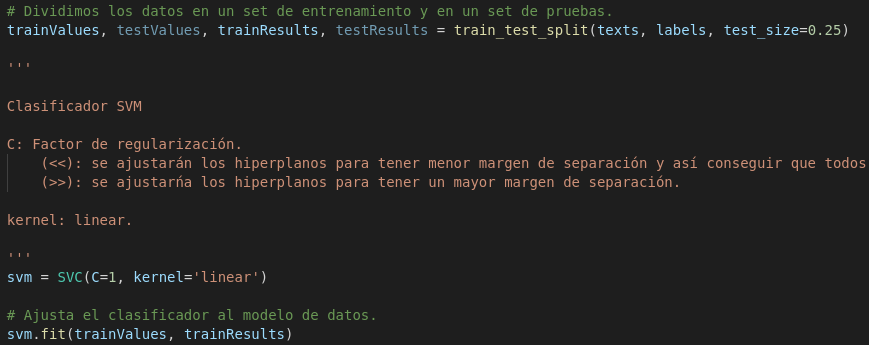
Ni → Nº de documentos en los que aparece el término clave i-ésimo.

* + - * Pesos TF - IDF: aparecen contenidos en los vectores del objeto TfidfVectorizer. Su cálculo es W ij = TF ij \* IDF i, y nos indica lo relevante que es una palabra para un documento en una colección.

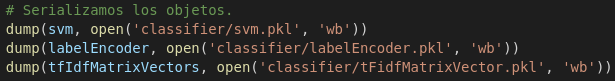
* + 1. Un objeto LabelEncoder (codificador de categorías) transforma los nombres de las clases de noticias (rugby, tenis, etc) a identificadores de tipo numérico.



* 1. Ahora que ya volvemos a contar con el codificador de categorías y los vectores TF - IDF de las noticias, vamos a utilizar estos para entrenar nuestra máquina de vector de soporte.
     1. En primer lugar, generamos un DataSet para entrenar la máquina de vectores de soporte, este se compone de un 25% de vectores TF-IDF para la realización de pruebas y un 75% para entrenar al modelo.
     2. Después de generar el DataSet, creamos una máquina de vectores de soporte configurada con los siguientes hiperparámetros:  
        + C = 1 → Supone la minimización de los márgenes, es decir, una reducción de la separación de los hiperplanos, el objetivo de esto es tratar de obtener el menor error posible a la hora de realizar una clasificación.
        + kernel = lineal → Por lo general, para la clasificación de textos puede utilizarse un kernel lineal, por eso lo hemos utilizado.
     3. Finalmente, entrenamos el modelo con el 75% de los datos.



* 1. Una vez que ya contamos de nuevo con la SVM, los vectores TF - IDF y el codificador de categorías, los serializamos para evitar que tener que volver a repetir estos 8 pasos y agilizar el proceso de carga.

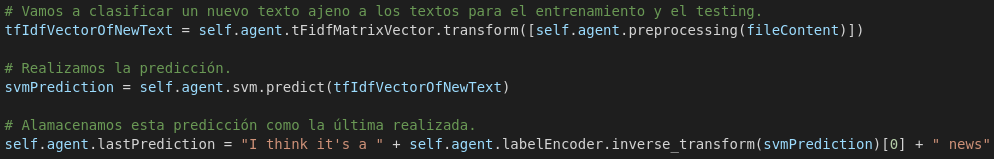


1. Cargamos en el agente la SVM, los vectores TF - IDF y el codificador de categorías.



Las acciones que realizaría este agente para la clasificación de noticias serían:

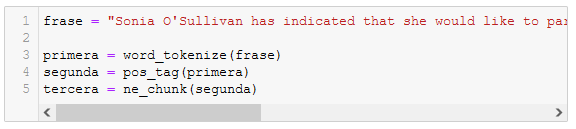
1. Una vez que le es pasado a este agente el nombre del fichero que contiene la noticia y su contenido es cargado en una variable, el texto se preprocesa para quedarnos únicamente con las palabras que no son stopwords y son alfabéticas, tal y como se describió antes. Después se utiliza esta noticia preprocesada y se obtiene su vector TF - IDF correspondiente.
2. Utilizamos el vector TF - IDF de la nueva noticia para pasárselo a nuestro modelo de SVM y que así este la clasifique en una categoría codificada.
3. Utilizamos el objeto LabelEncoder para transformar la categoría codificada en el nombre que esta tenía originalmente.
4. Posteriormente, la frase que indica el tipo de noticia es enviada al agente ChatBot para que este envíe la información hacia arriba en la arquitectura, para que así llegue a la vista y el usuario pueda ver el resultado de la clasificación.



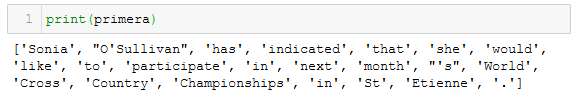
### Analizador

Este agente es el encargado de analizar un texto. Para realizar el análisis, el agente recibe un diccionario, el cual contiene el nombre del archivo de la noticia y las categorías que el usuario quiere conocer de la noticia dada haciendo uso de la librería NLTK.

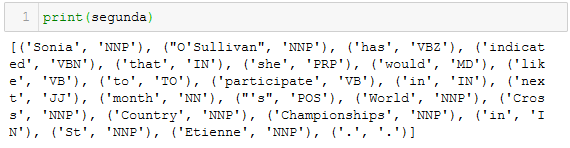
Para poder extraer la información deseada por el usuario, es necesario realizar unos pasos para la conversión del texto plano a una que pueda utilizar la librería.



La primera iteración consiste en “tokenizar” la frase, es decir, dividir el string de entrada en palabras. Quedando de esta manera el texto:



Una vez hemos dividido el texto, el siguiente paso es asignar etiquetas, el llamado etiquetado gramatical, el cual consiste en marcar una palabra en un texto como correspondiente a una parte particular del discurso, según su definición y su contexto. De tal manera que tendríamos una lista con tuplas de palabra y etiqueta.



Para finalizar, la técnica básica que utilizaremos para la detección de entidades es el chunking, que segmenta y etiqueta secuencias de varios tokens. NLTK proporciona un clasificador que ya ha sido entrenado para reconocer entidades con nombre, al que se accede con la función nltk.ne\_chunk(). Si establecemos el parámetro binary=True [1], las entidades con nombre sólo se etiquetan como NE; en caso contrario, el clasificador añade etiquetas de categoría como PERSON, ORGANIZATION y GPE.

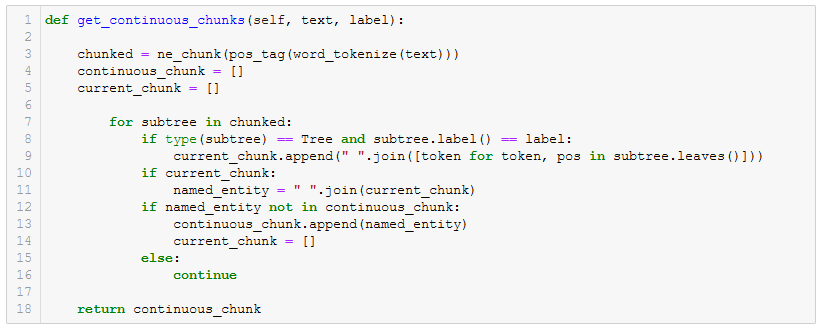


Una vez que hemos convertido el texto de la noticia, comenzamos el análisis del texto, comparando las etiquetas con las categorías que quiere conocer el usuario.

Permite elegir una o varias categorías entre las disponibles:

* Organization: organización que se menciona en la noticia.
* Person: personas que aparezcan en la noticia.
* Location: localización en la que ocurre la noticia.
* Date: fecha de la noticia.
* Time: de cuando es la noticia.
* Money: dinero en el caso de que se menciona en la noticia.
* Percent: porcentajes de los que se habla en la noticia.
* Facility: instalaciones que se mencionan en la noticia.
* GPE: lugar geopolítico donde se desarrolla la noticia.

Estas categorías son las disponibles, no implica que todas las noticias tengan valores para todas ellas, puede ser que no encuentre información sobre una categoría en la noticia.



Esto devuelve una lista de palabras de la categoría dada, en el caso de que no encuentre ninguna palabra sobre la etiqueta devolverá la lista vacía. Esta lista se recorre para formar el mensaje que contiene el resultado del análisis y que posteriormente será devuelto al agente ChatBot, el cual se encargará de mostrarlo por pantalla actualizando la vista.

### Recomendador

Para el sistema de recomendaciones se implementó un agente que se encarga de realizar la búsqueda de noticias sobre el tema que haya escogido el usuario. Tras hacer la búsqueda, la librería que usamos (GoogleNews) devolverá las noticias más recientes sobre dicho tema, estas noticias vienen con información adicional. Para este proyecto solo tenemos en cuenta los siguientes campos:

* *Title*: título de la noticia.
* *Desc*: pequeña descripción sobre la noticia.
* *Date*: el tiempo desde que se publicó la noticia hasta ahora.
* *Date time*: fecha en la que se publicó la noticia.
* *Link*: dirección web a la noticia.

Esta información se le devolverá al agente del ChatBot que será el encargado de mostrarlo por pantalla al usuario.En caso de que no se haya encontrado ninguna noticia devolverá un mensaje para informar al usuario.

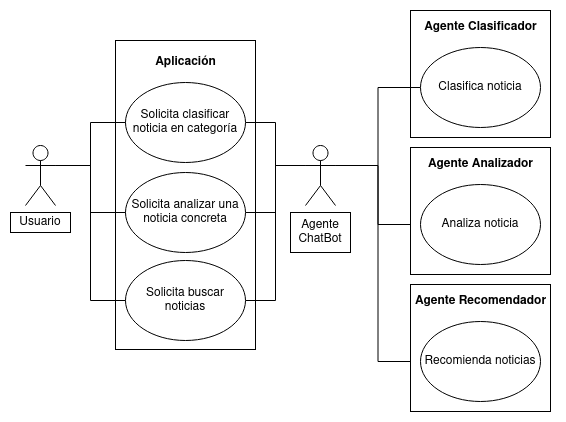
La búsqueda de noticias se hace bajo la siguiente configuración:

* Idioma: ya que las noticias que se están usando son todas en inglés, se ha limitado la búsqueda de nuevas noticias al mismo idioma.
* Tiempo: realiza una búsqueda de las noticias más recientes, para ser concretos, solo busca noticias que hayan sido publicadas en los últimos siete días.
* Codificación: se ha elegido el formato de codificación de caracteres de UTF-8.

Esta configuración se podría modificar o ampliar en caso de que fuera necesario.

## Arquitectura y diagramas

### Casos de uso

Este diagrama muestra las posibles interacciones del usuario con el sistema, también otros subsistemas que componen la aplicación y las acciones que realizan.  
  


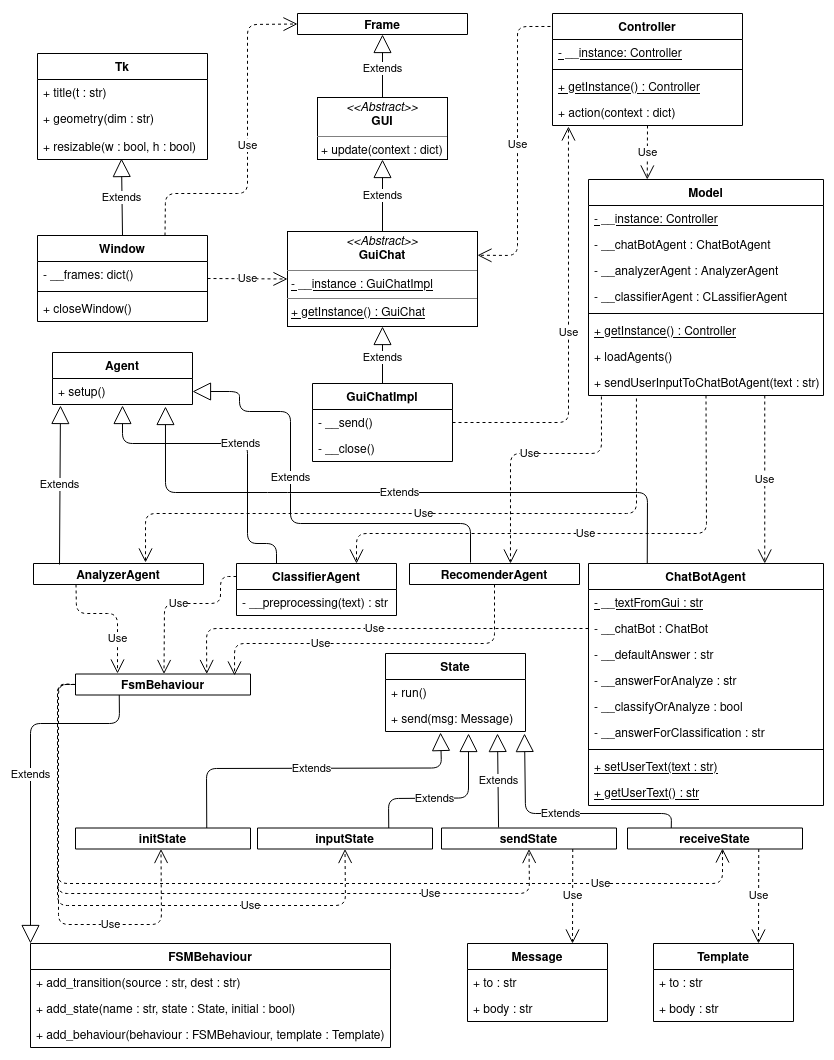
Las siguientes tablas detallan los casos de uso:

|  |  |
| --- | --- |
| **Caso de uso (ID)** | Solicitar clasificar una noticia (CU1) |
| **Objetivo en contexto** | El usuario solicita la clasificación de una noticia en el sistema. |
| **Entradas** | Nombre del fichero que contiene la noticia (texto plano). |
| **Precondiciones** | No procede. |
| **Salidas** | Categoría a la que pertenece la noticia. |
| **Postcondición si éxito** | No procede |
| **Postcondición si fallo** | No procede |
| **Actores** | Usuario |
| **Secuencia normal** | 1. El usuario solicita clasificación de una noticia. 2. El sistema pide el nombre del fichero que contiene la noticia. 3. El usuario indica el nombre del fichero que contiene la noticia. 4. El sistema clasifica la noticia. 5. El sistema muestra la categoría en la que se ha clasificado la noticia. |
| **Secuencia alternativa** | 4. El sistema indica que no se ha podido encontrar la noticia con el nombre especificado. |

|  |  |
| --- | --- |
| **Caso de uso (ID)** | Solicitar información de una noticia (CU2) |
| **Objetivo en contexto** | El usuario solicita extraer información de una noticia. |
| **Entradas** | 1. Nombre del fichero que contiene la noticia (texto plano). 2. Lista de las categorías de la información a extraer. |
| **Precondiciones** | No procede. |
| **Salidas** | Información sobre la noticia. |
| **Postcondición si éxito** | No procede |
| **Postcondición si fallo** | No procede |
| **Actores** | Usuario |
| **Secuencia normal** | 1. El usuario solicita extraer información de una noticia. 2. El sistema pide el nombre del fichero que contiene la noticia. 3. El usuario indica el nombre del fichero que contiene la noticia a analizar. 4. El sistema informa de las categorías de la información que se pueden extraer. 5. El usuario introduce las categorías de la información que desea ver separándolas mediante espacios. 6. El sistema analiza la noticia. 7. El sistema muestra la información sobre la noticia. |
| **Secuencia alternativa** | 6. El sistema indica que no se ha encontrado la noticia.  7. El sistema indica que no ha encontrado cierta categoría de información al analizar la noticia. |

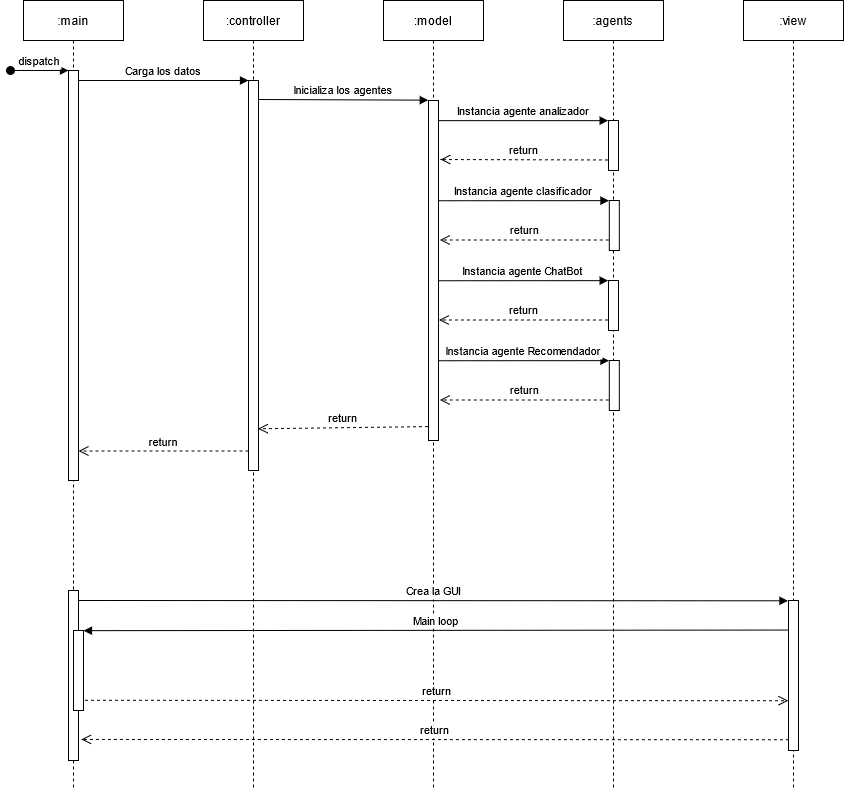
|  |  |
| --- | --- |
| **Caso de uso (ID)** | Solicitar una noticia (CU3) |
| **Objetivo en contexto** | El usuario solicita buscar noticias sobre un tema. |
| **Entradas** | Palabras clave para la búsqueda de la noticia. |
| **Precondiciones** | No procede. |
| **Salidas** | Listado de noticias relacionadas con las palabras clave introducidas por el usuario. |
| **Postcondición si éxito** | No procede |
| **Postcondición si fallo** | No procede |
| **Actores** | Usuario |
| **Secuencia normal** | 1. El usuario solicita noticias relacionadas con las palabras clave que este introduce. 2. El sistema pregunta las palabras clave para realizar la búsqueda. 3. El usuario indica la categoría. 4. El sistema busca noticias con dicha categoría. 5. El sistema muestra una lista de noticias, cada una de las cuales tiene asociada la siguiente información:    1. Título de la noticia.    2. Breve descripción de la noticia.    3. Fecha desde que se publicó la noticia.    4. Fecha en la que se publicó la noticia.    5. Enlace a la noticia. |
| **Secuencia alternativa** | 5. El sistema indica que no hay noticias relacionadas con las palabras clave introducidas. |

### Diagrama arquitectónico

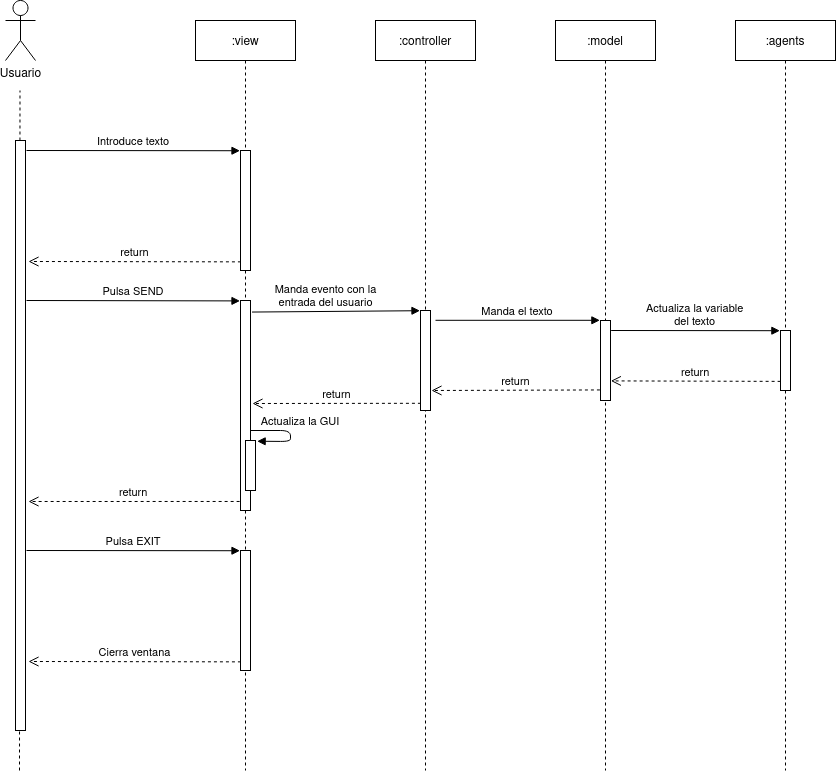


### Diagramas de secuencia

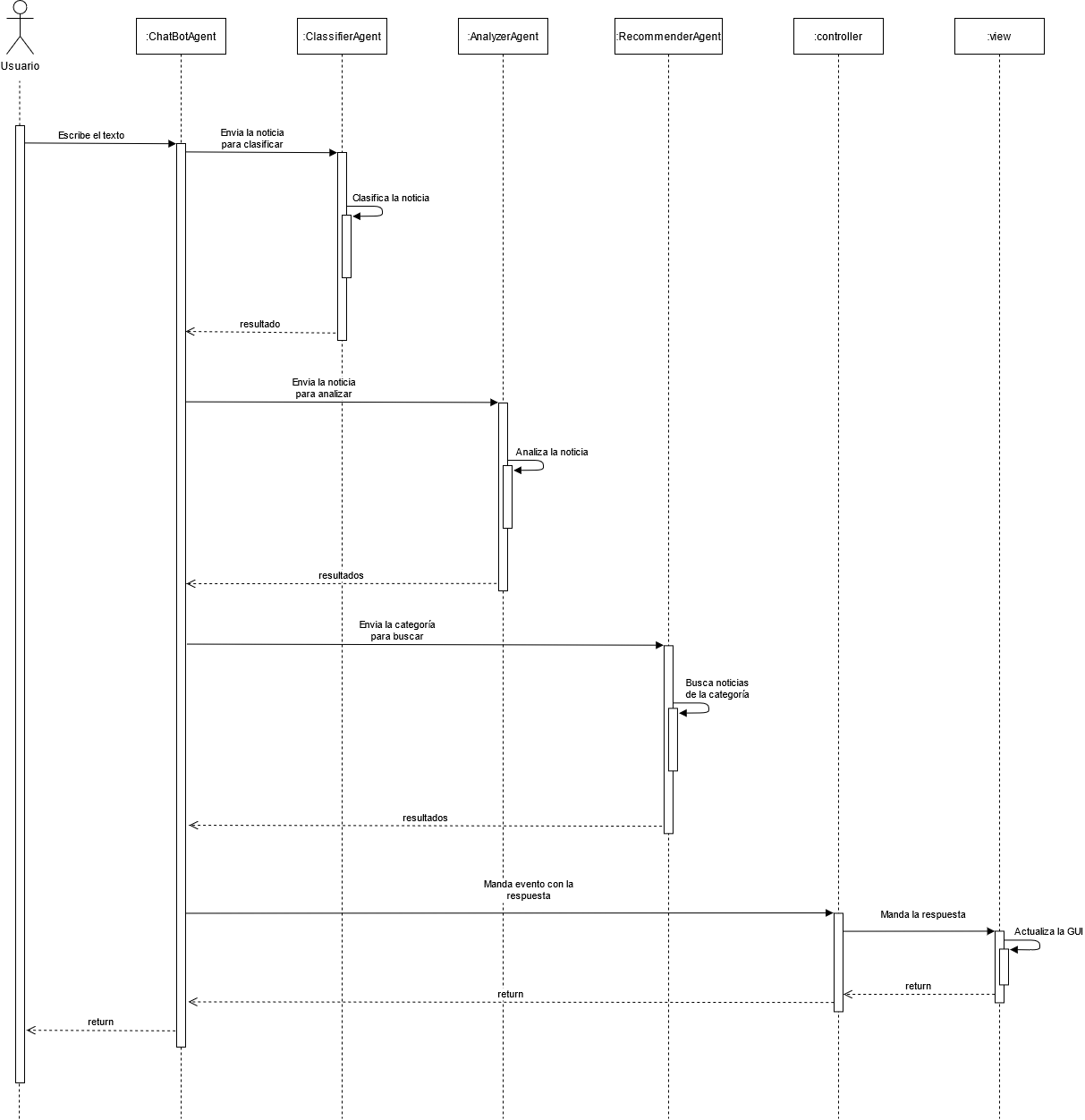
Este diagrama define el arranque y la configuración inicial del sistema software.



El siguiente diagrama representa gráficamente el flujo del programa que tiene lugar desde que el usuario introduce datos por la vista, baja por todos los niveles de la aplicación hasta los agentes y para finalizar vuelve a subir hasta la vista.



El diagrama que se muestra a continuación sirve para representar la comunicación entre los agentes del sistema y como una vez obtenida la información esta es devuelta a la vista, actualizando la interfaz gráfica.



# 

# Implementación

El lenguaje de programación empleado en nuestro proyecto ha sido Python en su versión 3.8, aunque también se admiten versiones superiores a esta.

Librerías utilizadas:

## Spade

* Características principales:  
  + Es una plataforma web de software libre (LGPL) creada en 2005 por la Universidad Politécnica de Valencia que permite el uso de sistemas multi-agente.
  + Sigue los estándares FIPA y XMPP.
  + Permite una comunicación P2P entre agentes.
  + Soporta los comportamientos: Cyclic, Periodic, Timeout, OneShot y FSM.
  + Soporta la comunicación con otras plataformas, como por ejemplo JADE, vía HTTP o XMPP.
  + Librería pensada para el lenguaje Python.
* Uso en el proyecto de DASI:
  + Han sido dados de alta 4 agentes en esta plataforma:
    - dasi1 (ChatBotAgent): Centrado en la gestión del ChatBot y en comunicarse con los otros agentes.
    - dasi2 (ClassifierAgent): Dedicado a la clasificación de las noticias deportivas.
    - dasi3 (AnalyzerAgent): Se ocupa de analizar las noticias, es decir, de extraer características.
    - dasi3 (RecomenderAgent): Encargado de realizar una búsqueda en la web sobre un tema elegido por el usuario.
    - Los 4 agentes anteriores han sido configurados como si fueran máquinas de estados finitos.
  + La comunicación interna de la aplicación es la siguiente:
    - Entre la aplicación News App y el agente ChatBot.
    - Entre el agente ChatBot y el agente clasificador.
    - Entre el agente ChatBot y el agente analizador.
    - Entre el agente ChatBot y el agente recomendador.

## ChatterBot

* Características principales:
  + Motor de diálogo basado en aprendizaje automático.
    - Funcionamiento conceptual: cuando un usuario introduce una frase el Bot guarda el texto que ha introducido el usuario, por lo que a medida que este recibe más entradas, aumenta el número de respuestas que puede dar y también la precisión de la respuesta. Esta se elige en función de la entrada conocida que más se aproxime y de la respuesta más probable, es decir, aquella emitida con más frecuencia por parte de un usuario.
  + Puede ser entrenado para hablar cualquier idioma.
  + Puede valerse de corpus ya creados, por ejemplo, para llevar a cabo conversaciones comunes con el ChatBot.
  + Puede configurarse para dar respuestas en base a colecciones de conversaciones conocidas.
    - Las conversaciones se representan como un grafo.
  + Librería pensada para el lenguaje Python.
* Uso en el proyecto de DASI:
  + El agente ChatBot termina recibiendo aquella información que se originó en la interfaz de usuario. Una vez que esta le llega, la analiza para emitir 1 de 3 salidas posibles, en función de esta, se sabrá si la acción a realizar es la de clasificar o analizar, o bien si hay que informar al usuario de que no se ha entendido la entrada introducida.

## Scikit-learn

* Características principales:
  + Biblioteca de software libre (BSD) para realizar tareas de aprendizaje automático.
  + Incluye varios algoritmos de clasificación, regresión, análisis de grupos (máquinas de vectores de soporte, random forest, k-means, dbscan, etc).
  + Compatible con otras librerías de Python como NumPy y SciPy.
  + Comunidad muy activa y grande; mucha documentación.
  + Librería pensada para el lenguaje Python.
* Uso en el proyecto de DASI:
  + Hemos empleado una máquina de vectores de soporte con un kernel lineal para realizar la clasificación de noticias en alguna de las categorías reconocidas.
  + También tiene su utilidad para el cálculo de los vectores TF - IDF de los textos de las noticias, que nos dicen para cada palabra lo relevante que es esta para un documento perteneciente a una colección concreta, por lo que nos servirán junto con la SVM para averiguar la pertenencia de una noticia a alguna de las categorías.
  + Otros usos que se le han dado a esta librería han sido:
    - Para la división automática del dataset de noticias en sets para el entrenamiento y testing.
    - Codificación de las categorías deportivas (etiquetas de texto) en etiquetas numéricas.

## 

## NLTK

* Características principales:
  + Conjunto de bibliotecas y programas para el procesamiento de lenguaje natural (PLN).
  + Permite varios idiomas.
  + Amplia comunidad y documentación.
  + Para programación en el lenguaje Python.
* Uso en el proyecto de DASI:
  + Se hace uso de esta tecnología para la extracción de información de una noticia. Es decir, poder analizar el texto y obtener los datos más relevantes de la noticia, como por ejemplo, deportistas, equipos o los lugares en los que se han jugado los partidos.

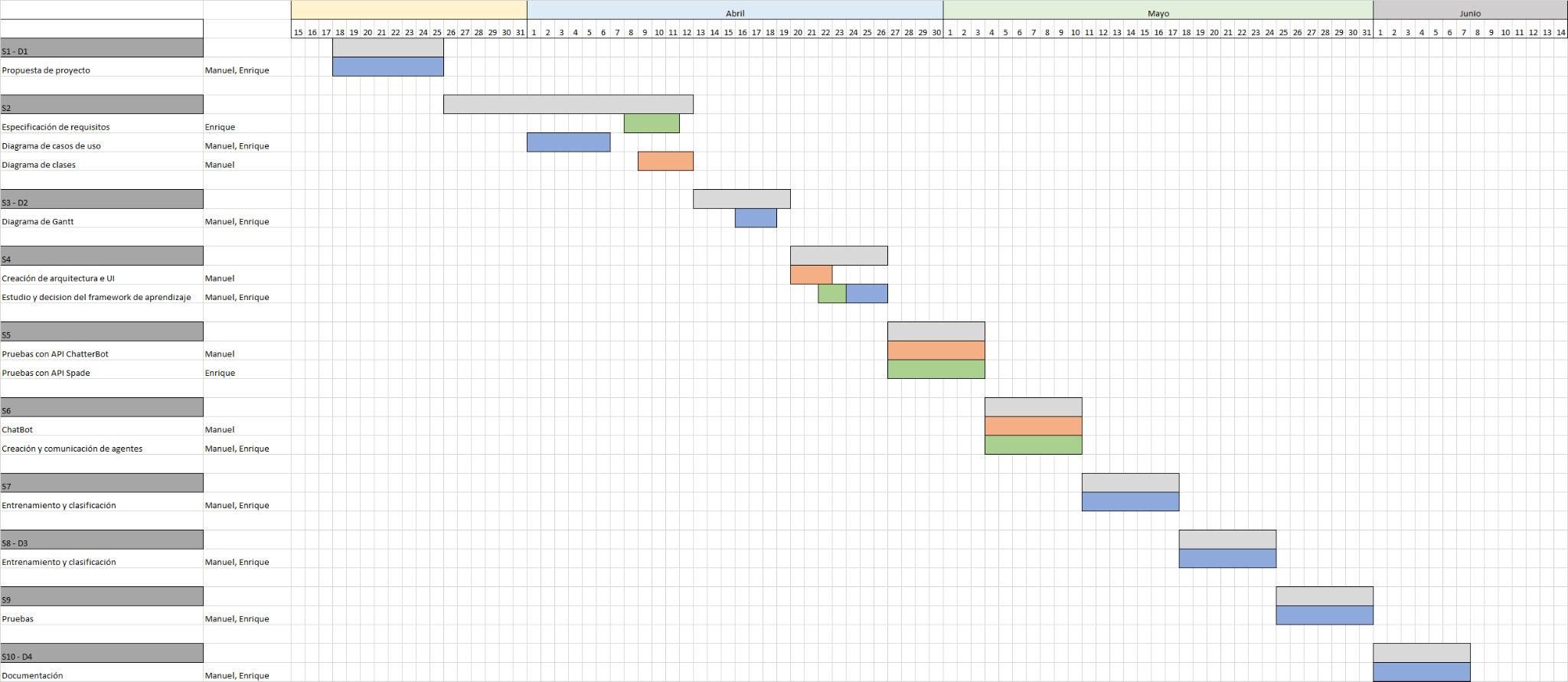
## GoogleNews

* Características principales:
  + Permite buscar noticias.
  + Extraer información de estas noticias, como por ejemplo, la fecha de publicación, la descripción o el link.
  + Fácil de usar.
  + El lenguaje de programación es Python.
* Uso en el proyecto de DASI:
  + Hemos implementado esta tecnología en el proyecto para el agente recomendador, el cual se encarga de realizar la búsqueda de noticias del tema elegido por el usuario.

# 

# Planificación

En cuanto a la dinámica de trabajo del proyecto en parejas hemos seguido la planificación del diagrama de Gantt que realizamos al comienzo del desarrollo.



Hemos trabajado usando GitHub para el control de versiones, y el fichero de excel con la planificación para la gestión de las tareas, en este último hemos ido detallando lo que cada miembro de este grupo hacía de cara al desarrollo del proyecto a lo largo de cada semana.

# Trabajo futuro

Actualmente el proyecto está orientado para clasificar, analizar y recomendar noticias deportivas. Actualmente distinguimos entre fútbol, rugby, cricket, atletismo y tenis, uno de los trabajos a futuro que se podría hacer es introducir nuevas categorías deportivas permitiendo clasificar entre más tipos de deporte.

Por otro lado, se podría enfocar el ChatBot para que en lugar de centrarse solo en noticias deportivas se ampliará para poder clasificar cualquier tipo de noticias, ya sea de política, de actualidad o de entretenimiento. Para realizar esto habría que reentrenar el modelo para que pudiera distinguir las nuevas noticias, en cambio, para el analizador de noticias y el recomendador no habría que realizar cambios grandes, ya que se centran en sacar la información de las categorías que pase el usuario y de buscar lo que el usuario quiera.

Otro de los retos sería la investigación de otros kernels de SVM para la clasificación de las noticias, pero también la experimentación con otros clasificadores para poder realizar un poco comparativa entre ellos, por ejemplo, intentarlo clasificación con una Red Neuronal Convolucional o algún algoritmo de clustering.

Para finalizar, se podría mejorar la interfaz del ChatBot haciéndola más atractiva.

# Conclusiones

Es posible construir una arquitectura software, por ejemplo MVC como en este caso, e integrarla correctamente haciendo uso de otros paradigmas de programación como es la programación con agentes. El sistema es modular y escalable, por lo que, en caso de ser necesario, se podrían añadir nuevas funcionalidades o agentes sin necesidad de modificar mucho el código ya existente.

Si bien es cierto que en la asignatura de Desarrollo de Aplicaciones y Sistemas Inteligentes hemos trabajado con agentes basados en Java, para este proyecto hemos optado por el lenguaje de Python, ya que consideramos que hay una mayor cantidad de librerías enfocadas tanto a la inteligencia artificial como a procesamiento de lenguaje natural.

Después de investigar los clasificadores adecuados para el problema de clasificación realizando una ámplia búsqueda, encontramos una amplia comunidad detrás del uso de las SVM’s, uno de los modelos de inteligencia artificial más extendidos y recomendados para la clasificación de texto.

Ya contábamos con algo de experiencia a la hora de realizar la clasificación y el análisis de textos, ya que cursamos la asignatura de Sistemas de Gestión de la Información durante el primer cuatrimestre, donde aprendimos a desarrollar índices de búsqueda y a trabajar con vectores TF - IDF, por lo que estos puntos no supusieron una gran dificultad.

# Bibliografía

Documentación de Spade: <https://spade-mas.readthedocs.io/en/latest/>

Documentación de ChatterBot: <https://chatterbot.readthedocs.io/en/stable/>

Documentación de Scikit-Learn: [https://scikit-learn.org/stable/#](https://scikit-learn.org/stable/)

Documentación NLTK: <https://www.nltk.org/>

Documentación GoogleNews: <https://github.com/Iceloof/GoogleNews>

Arlow, J., Neustadt, I. (2006). UML 2, Anaya Multimedia, Madrid.

# Anexo 1: Manual de instalación

# Introducción

Este documento detalla los pasos a seguir para realizar la instalación, ejecución y desinstalación de la aplicación del proyecto de DASI.

AVISO:

* La instalación que se describe a continuación es funcional solamente en el sistema operativo Ubuntu, derivadas de Ubuntu o máquinas virtuales de este.
* Durante el desarrollo del proyecto hemos trabajado con python en sus versiones 3.7 y 3.8.

# Instalación

Para instalar la aplicación, realizar los siguientes pasos:

1. Descargarse el proyecto de GitHub ejecutando en la terminal del sistema:  
     
   git clone <https://github.com/ManuelGMS/DASI>
2. Una vez descargado, acceder a la carpeta “inst-uinst”.
3. Ejecutar la siguiente instrucción para instalar las librerías: “bash install.sh”.

AVISO IMPORTANTE:

* La librería “chatterbot” suele ser problemática:
  + Si observa que su instalación llega a los 10 minutos, aborte su ejecución con CTRL + C, no importa lanzar esta interrupción por teclado ya que es la última librería que se instala. Después de esto, lanzar el comando: “python3 -m pip install chatterbot” y comprobar que llega a instalarse del todo.

* + Si la ejecución de la instalación de la librería “chatterbot” es abortada, lanzar el comando: “python3 -m pip install chatterbot”. Después de lanzarlo, comprobar que la librería llega a instalarse del todo, pues es la última librería necesaria para poder ejecutar la aplicación.
* Compruebe que puede ejecutarse la aplicación según lo especificado en el apartado de “Ejecución”.

# Ejecución

Para ejecutar la aplicación, seguir los siguientes pasos:

1. Abrir una terminal del sistema y posicionarse en la carpeta “DASI”.
2. Ejecutar el comando “bash launcher.sh”, este realizará algunos ajustes y levantará la aplicación, y una vez que eso ocurra, ya podrá utilizarse.

# Desinstalación

Para desinstalar la aplicación, seguir los siguientes pasos:

1. Acceder a la carpeta “inst-uinst”.
2. Una vez allí, ejecutar la siguiente instrucción: “bash uninstall.sh”.

Este script nos preguntará qué librerías del sistema queremos desinstalar y cuáles no. Las librerías de python podrían reutilizarse en otros proyectos y por tanto no se eliminarán.

# Ficheros de código fuente

* Carpeta “inst-uinst”:
  + install.sh: este fichero es para la instalación de las librerías del sistema y de python que son necesarias para la ejecución de la aplicación.
  + uninstall.sh: fichero que contiene un script que pregunta al usuario que librerías del sistema instaladas por el proyecto se desea desinstalar.
* Carpeta “DASI”:
  + launcher.sh: este script configura algunas rutas que deben de estar establecidas antes de lanzar la aplicación mediante la invocación del fichero main.py, de lo contrario, algunas librerías de la aplicación no se encontrarán y por tanto, obtendríamos errores.
  + main.py: fichero que inicializa la aplicación.
  + model.py: modelo de la aplicación, realiza tareas como la carga de los agentes, que son considerados como el nivel de acceso a datos, y también permite pasar la información del usuario que llega por el controlador al agente ChatBot, el cual se ocupará de actuar o informar al agente que corresponda para que realice una tarea.
  + view.py: contiene la ventana y la vista (*Frame*) de la aplicación. Desde este nivel, se propagan mensajes hacia el controlador con información del usuario.
  + controller.py: núcleo de la arquitectura MVC, gestiona todos los eventos que genera la aplicación y orquesta las acciones pertinentes en base a estos.
  + chatBotAgent.py: código fuente del agente ChatBot.
  + classifierAgent.py: código fuente del agente Clasificador.
  + analyzerAgent.py: código fuente del agente Analizador.
  + recomenderAgent.py: código fuente del agente Recomendador.

# Ficheros de recursos

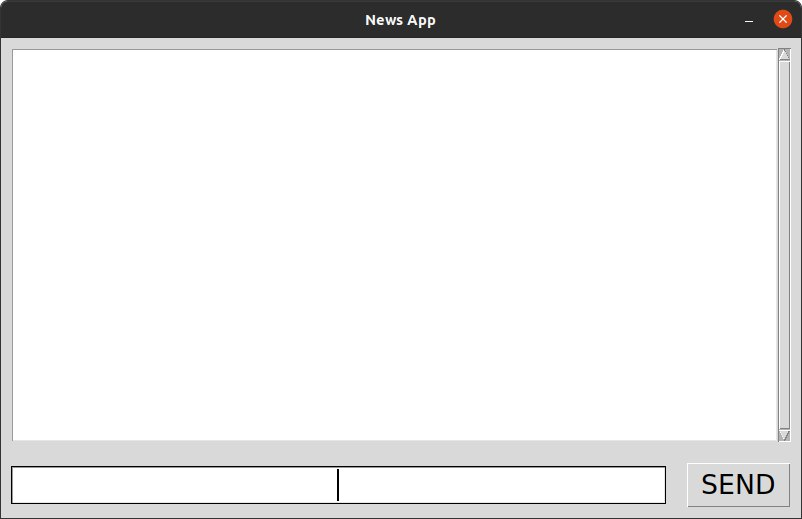
* Carpeta “bbcsport”:
  + Contiene varias subcarpetas de categorías deportivas, las cuales contienen a su vez ficheros de texto plano con noticias relativas a dicha categoría.
    - AVISO: se pueden introducir nuevas noticias en las subcarpetas, pero lo que no se debe de hacer es eliminar esta carpeta o desbalancear en exceso la cantidad de noticias en estas subcarpetas que reflejan las categorías deportivas conocidas.
* Carpeta “chatterbot”:
  + Fichero “database.sqlite3”: contiene información relativa a lo aprendido por el ChatBot con las conversaciones utilizadas en su proceso de entrenamiento.
    - AVISO: su borrado implica que se desea reentrenar el ChatBot otra vez con el fichero de “corpus.json”.
  + Fichero “corpus.json”: contiene las conversaciones con las que entrenar al ChatBot para realizar su tarea.
    - AVISO: se pueden crear nuevas frases de entrada, pero no se deben inventar nuevas frases de salida que no estén ya en el fichero.
* Carpeta “classifier”:
  + newsClassified.csv: un fichero de valores separados por comas que contiene dos columnas, en la primera están los textos de cada noticia y en la segunda la categoría deportiva a la que pertenece cada noticia.
    - AVISO: la eliminación de este fichero indica el reentrenamiento de la máquina de vector de soporte con las noticias del dataset “bbcsport”, por lo que los ficheros de tipo \*.pkl se borrarán y se volverán a generar automáticamente, este fichero también se regenerará de forma automática si es eliminado.
  + svm.pkl: contiene la serialización del clasificador de la aplicación, este está basado en una máquina de vectores de soporte.
    - AVISO: este fichero no se debe modificar o eliminar manualmente.
  + labelEncoder.pkl: es un objeto serializado que contiene las etiquetas de las clases (categorías deportivas) codificadas con identificadores numéricos que van de 0 a (N - 1) clases.
    - AVISO: este fichero no se debe modificar o eliminar manualmente.
  + tFidfMatrixVector.pkl: este objeto serializado es una matriz de vectores TF - IDF asociados al texto de cada noticia del dataset “bbcsport”.
    - AVISO: este fichero no se debe modificar o eliminar manualmente.
* Carpeta “nltk\_data”:
  + Contiene librerías adicionales de NLTK.
    - AVISO: esta carpeta no debe de ser modificada ni eliminada.
* Carpeta “news”:
  + Es una carpeta en la que los agentes de clasificación y análisis buscan un fichero con un cierto nombre para realizar sus respectivas tareas.
    - AVISO: es posible añadir o eliminar noticias de la carpeta, pero la carpeta no debe de ser eliminada.
* Carpeta “docs”:
  + Contiene la licencia de software del proyecto y el fichero de readme.
    - AVISO: no modificar los ficheros.
* Carpeta “inst-uinst”:
  + Contiene los scripts de instalación y desinstalación de la aplicación.
    - AVISO: no modificar los scripts sin conocimiento.

# 

# Anexo 2: Manual de usuario

## Ventana principal

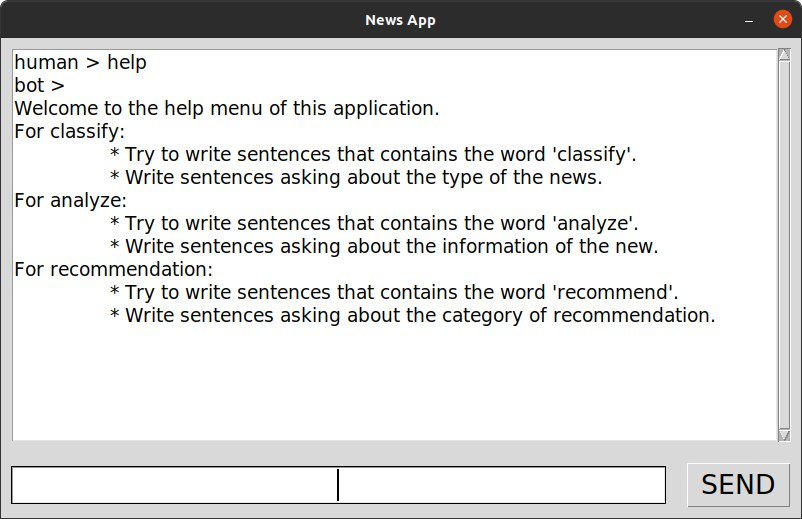
Al iniciar la aplicación aparece directamente el chat. En esta ventana podemos ver en la parte superior derecha los botones para minimizar y cerrar la aplicación. En el centro vemos el cuadro donde aparecerá la conversación. Por último en la parte inferior vemos el cuadro donde el usuario introducirá el texto y a la derecha un botón para enviar el mensaje.



El bot analizará el texto que introduzca el usuario, puede distinguir la opción de ayuda y las tres funcionalidades que presenta el proyecto (clasificar, analizar o recomendar).

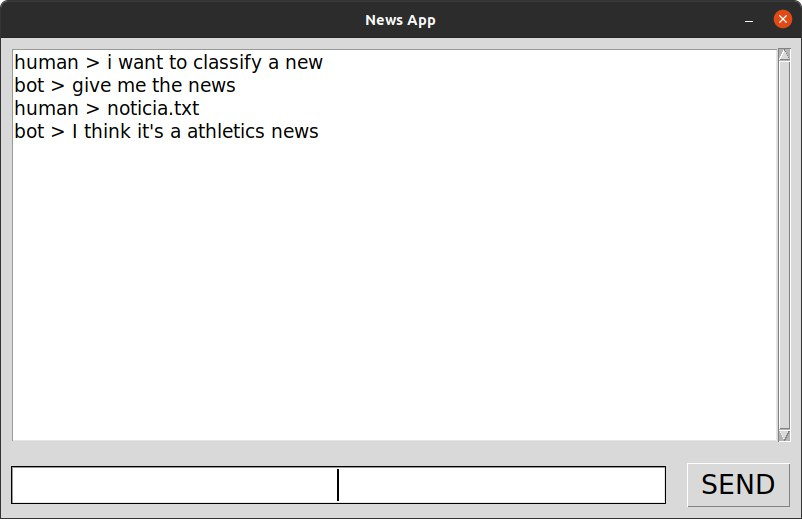
## Ayuda

En la siguiente captura se puede ver la opción de ayuda, si el ChatBot detecta que el usuario está pidiendo ayuda, le responde con un pequeño texto explicativo. En este texto informa al usuario de las funcionalidades que dispone, y de la entrada que espera para cada una de ellas, así como la información que esperara a continuación.



## Clasificador

Para que el ChatBot lance la funcionalidad de clasificar una noticia, espera recibir una entrada de texto que contenga la palabra “classify” o alguna frase en la que el usuario pida clasificar una noticia. A continuación, pedirá por el nombre de la noticia, el cual tendrá que estar ubicado en la carpeta *”news”*.Tras indicarle la noticia que deseas clasificar devolverá la categoría a la que se asemeja.



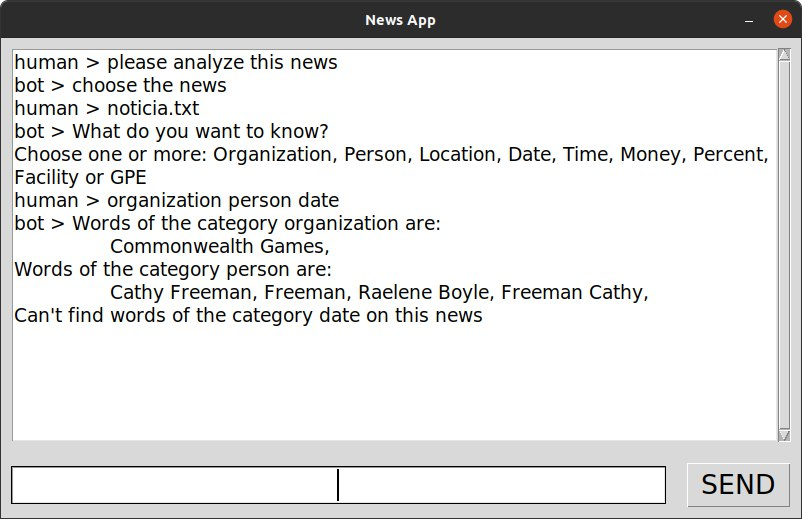
## Analizador

Pedirle al ChatBot que analice una noticia es similar a pedirle que clasifique, en lugar de introducir una entrada con la palabra “classify” o variantes, espera recibir “analyze”.

Después de esto te pedirá introducir el nombre de la noticia y por último seleccionar una o más categorías sobre las cuales extraerá la información.

El usuario podrá elegir entre, organización, persona, localización, fecha, tiempo, dinero, porcentaje, instalaciones o la entidad geopolítica (Organization, person, location, date, time, money, percent, facility o GPE).

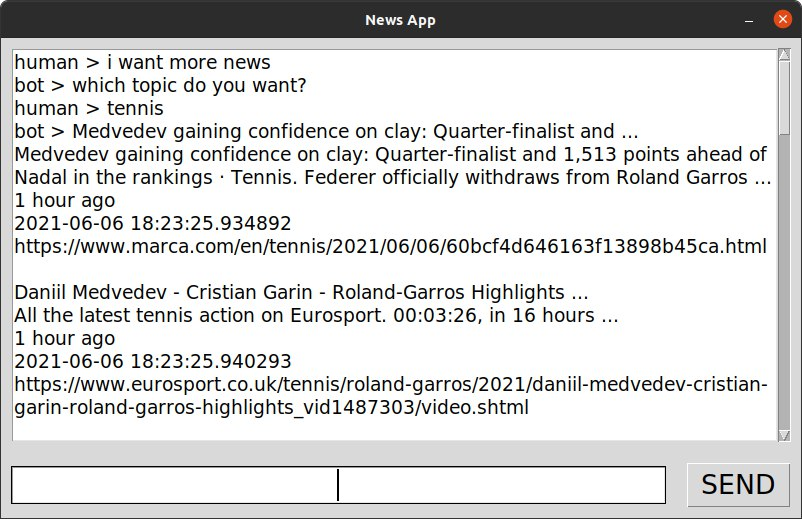
Tras seleccionar las características para que extraiga información, se mostrarán los resultados por pantalla, indicando las palabras que se han encontrado, o un mensaje de que no hay palabras para esa categoría.



## Recomendador

Para que recomienden más noticias, el ChatBot espera una entrada similar a “more news” o variantes que impliquen que desea conocer más noticias sobre un tema. Tras detectar esto, pedirá al usuario que introduzca el tema del cual quiere leer más noticias.

Tras esto, realizará una búsqueda en la web y devolverá por pantalla las noticias más recientes. Mostrando, tanto el título, una pequeña descripción, el tiempo que ha transcurrido desde su publicación, la fecha en la que se ha publicado y por último el link a la noticia (el cual se puede copiar para pegarlo en el navegador).



## Error

En el caso que no reconozca la entrada del usuario como ninguna de las funcionalidades, indicará al usuario que no entiende la entrada.

