Implementación de un bot agenda en Rasa

Trabajo final Reconocimiento Automático del Habla

Victoria Beltrán Domínguez

Enero 2022

Contents

1 Resumen

Este trabajo plantea construir un bot con funciones de agenda a través de rasa, la tecnología expuesta en las prácticas de esta asignatura. Este bot nos ayudará a manejar diferentes contactos, y también a programar ciertos recordatorios.

Así, esta memoria se desglosa en diferentes partes: en la primera parte se plantearán los objetivos establecidos para este proyecto, o dicho de otra manera, qué queremos conseguir. A continuación, una vez claros los objetivos, pasamos a detallar la implementación del bot. Seguidamente, se comentarán ciertas propuestas de mejoras para el bot. Finalmente, acabaremos con unas conclusiones.

2 Objetivos

Los objetivos propuestos en este trabajo son claros. Queremos construir un bot que permita tanto el manejo de contactos como la construcción de recordatorios. De esta manera, el bot deberá permitir no sólo añadir y borrar contactos, sino también recibir información general o particular de ellos. Además, como ya se ha dicho, especificando una hora y una descripción el bot deberá funcionar como una alarma y avisar de alguna manera, actuando así como una agenda viviente.

Todo ello por supuesto, en lenguaje natural, como si estuviéramos teniendo una conversación casual con nuestro bot.

Aparte de la implementación del bot, otro objetivo es experimentar con rasa, para ver hasta qué punto es capaz de crear un bot con cierta calidad. Cabe destacar, que en este proyecto no se contempla la implementación con rasa x, se quiere crear un bot utilizando sólo las herramientas provistas por rasa.

3 Implementación

Con el fin de explicar cómo se ha implementado el bot, vamos a desglosar este apartado en diversas partes. En la primera parte, se detallará no sólo la configuración de rasa, sino qué herramientas externas se van a utilizar. A continuación, se especificarán todos los *intents* definidos para el usuario, así como los diferentes formularios, slots y acciones perfilados en rasa. También se mostrarán qué reglas y stories se han definido, conectando así todas las secciones anteriores. Para acabar, se mostrarán ejemplos de funcionamiento del bot final.

3.1 Configuración rasa

Debemos definir qué secuencia de componentes procesarán los mensajes provenientes del usuario. Como queremos implementar nuestro bot en español, lo primero que haremos será definir el modelo de lenguaje preentrenado de noticias en español de SpacyNLP. Además, tokenizaremos, extraeremos embeddings y añadiremos también Duckling para poder extraer entidades. En nuestro caso, queremos que Duckling tenga en cuenta entidades de tipo tiempo, email y teléfonos. Además, también añadiremos el componente *DietClassifier* tanto para clasificar los mensajes en las intenciones definidas como para definir entidades propias y poder extraerlas.

En cuanto a las políticas definidas (aquellas que decidirán que acción tomar a continuación), se ha definido que se tengan en cuenta las reglas e historias establecidas, así como el contexto (conversaciones previas y slots completados). Esto lo hemos conseguido mediante *RulePolicy*, *MemoizationPolicy* y *TEDPolicy*.

Todos estos componentes, junto con los intents definidos, nos ayudarán a entender al usuario.

3.2 Intents

Como bien hemos dicho, necesitamos saber qué intenciones tiene el usuario. A parte de las que vienen por defecto (saludar, afirmar, negar,...), hemos definido ciertos intents para nuestro problema particular

- **programar**: es la intención que el usuario utilizará cuando quiera planificar una alarma.
- **registrar_contacto**: pretende representar la intención del usuario de añadir a un contacto.
- **dar_info**: representa la intención del usuario de contestar a las preguntas genéricas de añadir un contacto. Las respuestas contempladas son "Ponle [Carlos](entity)", "El teléfono es 642225579" o "el correo es lala@hotmail.es".
- **obtener_info**: cuando el usuario quiere recuperar la información de un contacto. Se contempla recuperar información total o parcial, pero siempre de un contacto.
- **borrar_contacto**: Simboliza la intención de borrar un contacto de la agenda.
- **obtener_todos_contactos**: Obtener todos los nick guardados como contactos.

Estas son todas las intenciones contempladas por parte del usuario.

3.3 Forms y slots

Se han definido tres formularios diferentes para poder sacar información del usuario. A su vez, para esos forms se han declarado una serie de slots con el fin de poder recordar toda la información que el usuario quiere registrar. De esta manera tenemos:

- **añadir_contacto_form**:

  slots:

  - **nick**: nombre con el que queremos guardar el contacto. Se han especificado muchos nicks en el conjunto de datos de entrenamiento, con el fin de que pueda ser reconocido automáticamente. Este puede ser inferido, pero esta es la entidad más difícil de detectar correctamente, pues el nombre de una persona puede ser cualquier cosa.
  - **email_amigo**: email del contacto que estamos añadiendo. No influencia la conversación y lo podemos relacionar con las entidades extaídas de tipo email de duckling.

  - **phone-number**: teléfono del contacto que estamos añadiendo. No influencia la conversación y lo podemos relacionar con las entidades extraídas de tipo phone-number de duckling.

- **añadir_evento_form**:

  slots:

  - **evento**: Descripción del evento a recordar.
  - **email_propio**: email donde quieres recibir el recordatorio. Nuevamente, lo podemos relacionar con las entidades extraídas de tipo email de duckling.
  - **alarma**: tiempo en el que queremos que nos llegue el recordatorio. No influencia la conversación y lo podemos relacionar con las entidades extraídas de tipo time de duckling.

- **borrar_contacto_form**:

  slots:

  - **nick_contacto**: Nick del contacto que pretendemos borrar. Se han especificado muchos nicks en el conjunto de datos de entrenamiento, con el fin de que pueda ser reconocido automáticamente. Este puede ser inferido, pero esta es la entidad más difícil de detectar correctamente, pues el nombre de una persona puede ser cualquier cosa.

3.4 Acciones

Hemos definido las siguientes acciones:

- **validate_añadir_evento_form**: esta acción se activará para validar los slots definidos para el evento_form. Debemos destacar que solamente se ha añadido validación para la alarma, para asegurarnos de que el formato recibido por duckling es el correcto, y también para validar que la fecha que se ha puesto de la alarma no está en el pasado o en el presente (pues no tiene sentido poner una alarma ahora, pero sí en un minuto).

- **programar_alarma**: está acción recoge los valores definidos en los diferentes slots relacionados con el añadir_evento_form, y programa un email para la hora definida. Para programar un evento asíncrono, se ha utilizado la librería *apscheduler*. Además, para enviar el correo, se ha utilizado la librería *smtplib*, con un correo creado especialmente para el bot en la plataforma *GMX*.

- **action_guardar_contacto**: recogiendo los valores de los slots relacionados con el añadir_contacto_form, guarda el contacto en un json.

- **action_borrar_contacto**: busca el nick especificado y si lo encuentra, lo borra de la agenda.

- **action_dar_info**: busca el nick especificado y si lo encuentra, devuelve la información pedida de este. Se puede pedir el teléfono, el email o información general.

- **action_obtener_todos_contactos**: acción que devuelve todos los nicks guardados en la agenda.

- **reset**: borra todos los valores de los slots. Esta acción es ejecutada cuando acabamos cualquier tarea.

3.5 Rules y stories

Se han definido una serie de reglas para poder predecir qué acción tomar en cada momento:

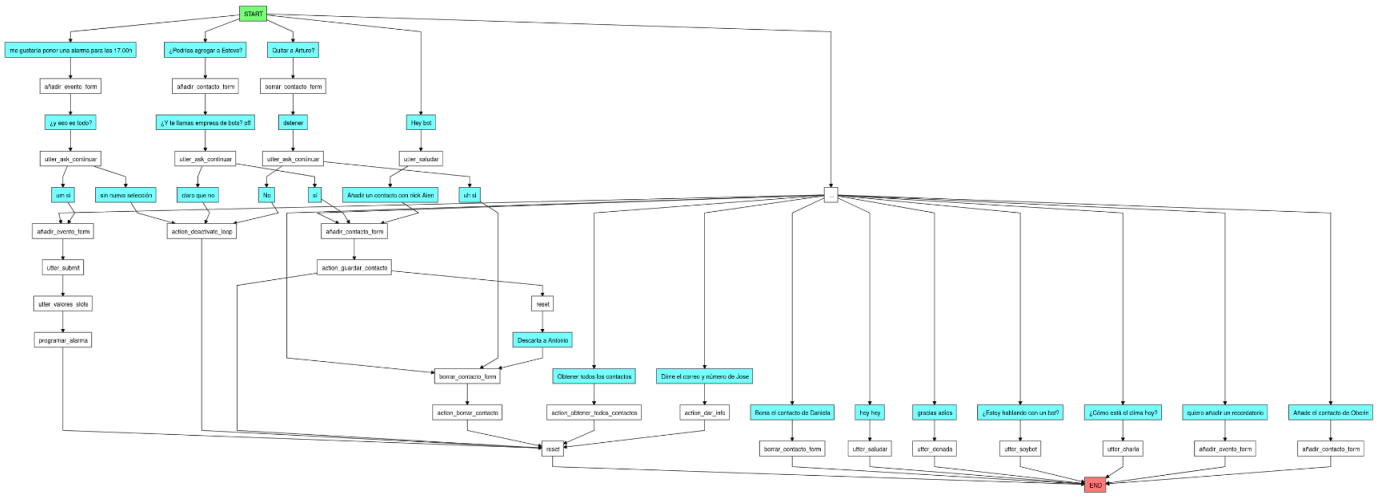
- **activar evento form**: cuando la intención del usuario es programar el evento, se activa el formulario para preguntarle los slots necesarios.

- **enviar evento form**: una vez hemos acabado de rellenar el formulario, entregamos los valores, los ponemos por pantalla y activamos la acción para programar la alarma.

- **activar contacto form**: cuando la intención del usuario es añadir una contacto, se activa el formulario para preguntarle los slots necesarios.

- **enviar contacto form**: una vez hemos acabado de rellenar el formulario, activamos la acción de guardar_persona.
- **obtener información sobre una contacto**: cuando la intención es obtener información sobre un usuario, activamos la acción dar_info.
- **activar formulario borrar contacto**: cuando la intención es borrar un contacto, activamos el formulario borrar_contacto.
- **activar acción de borrar contacto**: una vez hemos acabado de rellenar el formulario, activamos la acción borrar_contacto.
- **obtener información sobre contactos guardados**: activar la acción correspondiente cuando la intención del usuario es saber qué contactos tiene.

También se han definido una serie de stories para especificar el flujo de los formularios (cuál sería el flujo normal y qué hacer en caso de que el usuario quiere parar). Si ejecutamos el rasa visualize, obtenemos el siguiente grafo:



3.6 Ejemplos de funcionamiento del bot

En esta sección simplemente se van a mostrar capturas del bot-agenda para poder apreciar la funcionalidad conseguida.

Ejemplo de bot añadiendo contacto

```
Your input ->  hola
¡Hola! Soy un asistente para la organización. Puedes programar recordatorios y manejar contactos.
¿Cómo puedo ayudarte?
Your input ->  quiero añadir un contacto
¿Cómo guardo a este contacto?
Your input ->  Ponle Carlos
¿Email del contacto?
Your input ->  carlos@gmail.com
¿Teléfono móvil?
Your input ->  645352515
Guardando nuevo contacto con:
        Nick:Carlos
        Email:carlos@gmail.com
        Teléfono:645352515
Todo perfecto!
```

Ejemplo del bot obteniendo información de un contacto

```
Your input ->  dame información del contacto Carlos
El email de Carlos es carlos@gmail.com y el teléfono es 645352515
Your input ->  dame el email de carlos
El email de carlos es carlos@gmail.com
Your input ->  dame el telefono de carlos
El teléfono de carlos es 645352515
Your input ->
```

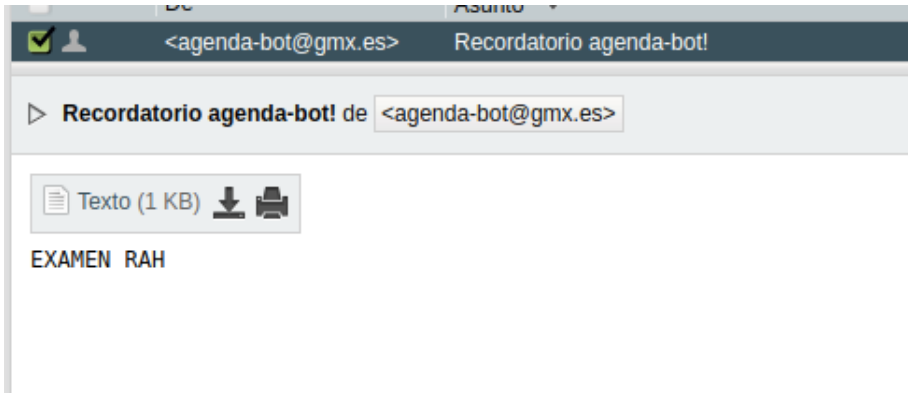
Ejemplo del bot borrando un contacto

```
Your input ->  borrar al contacto Carlos
Usuario Carlos borrado con éxito
Your input ->  telefono de Carlos
No tengo guardado el usuario Carlos
Your input ->
```

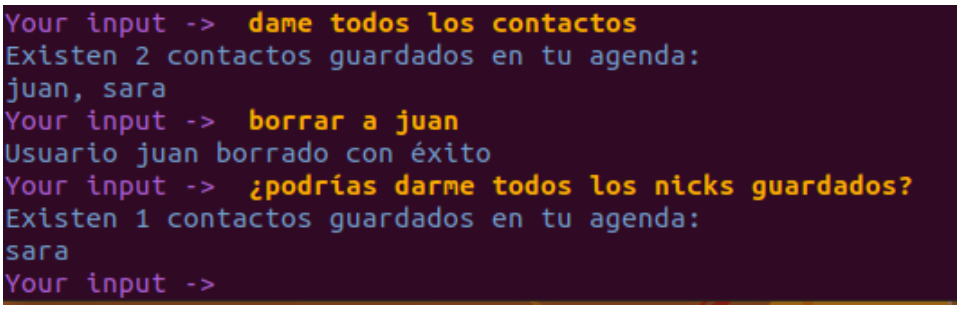
Ejemplo del bot añadiendo un evento

```
Your input ->  programaar un evento para una tarea
¿Cuál es el nombre del evento?
Your input ->  EXAMEN RAH
¿Cuál es tu email?
Your input ->  vicbeldo@inf.upv.es
¿Para cuándo quieres programar la alarma de este evento?
Your input ->  en un minuto
¡Listo!
Programo una alarma con las siguientes características:
 - Evento: EXAMEN RAH
 - Alarma: 30-01-2022 12:02
Your input ->
```

Ejemplo del bot enviando un recordatorio al correo title



Ejemplo del bot devolviendo todos los contactos



4 Futuras mejoras

Aunque el bot funcione, debemos hacer una retrospección sobre las dificultades encontradas en el desarrollo, y una propuesta sobre maneras de solventar los comportamientos incorrectos que podemos encontrar en el bot.

Una de las principales dificultades del bot ha sido el reconocimiento de entidades persona. Aunque pueda parecer un problema trivial para nosotros como humanos, ahora mismo este no tiene datos de entrenamientos suficientes para funcionar bien. Se ha intentado poner muchísimas entidades 'nick' para que el modelo pudiera distinguir bien, pero sigue sin ser suficiente. Como futura mejora, se podría integrar un componentes preentrenado a la pipeline para añadir funcionalidad a nuestro modelo.

Otra dificultad encontrada en el desarrollo ha sido la programación de un evento para un momento del futuro. A pesar de que en este proyecto se ha usado una librería de Python para ejecutar eventos asíncronos en conjunto con una para enviar correos, inicialmente se quería que fuera el bot por el mismo entorno por el que está hablando el que te lo recordara, sin necesidad de correo. Para ello, se encontró el método ReminderScheduled, que parecía hacer todo lo que queríamos lograr. Sin embargo, y después de muchos intentos fallidos, no se ha conseguido que este funcione como se deseaba. Más tarde se encontró que los recordatorios no funcionan en canales de solicitud-respuesta como el canal de rasa shell sino que había que utilizar un CallbackInput. Se deja esta funcionalidad como potencial mejora.

Finalmente, y como problema general, podemos destacar los datos de entrenamiento, pues han sido recogidos por mi, y no son representativos. Como trabajo futuro, se debería añadir más datos de entrenamiento con el fin de conseguir más certeza por parte del bot.

5 Conclusiones

En este trabajo se ha planteado la construcción de un bot-agenda. Se han especificado todos los componentes necesarios para poder construir el bot, así como sus conexiones y cómo conseguir comportamientos particulares de este. También se ha mostrado su funcionamiento y se han planteado algunas mejoras, que o bien por dificultad o bien por falta de tiempo, el alumno no ha sido capaz de realizar. Consideramos que los objetivos establecidos inicialmente han sido cumplidos con éxito.1