MANUAL DE RASA

INSTALACIÓN

Es el framework más popular a la hora de desarrollar asistentes conversacionales.

Para acceder a Rasa debemos descargarlo desde la página web oficial (rasa.com). Al tratarse de un framework de Python, es necesario que el ordenador disponga de un Python Environment Setup instalado.

Podemos descargar rasa desde la terminal del ordenador, escribiendo en ella “pip3 install RASA” y pulsando la tecla intro, lo cual activará la descarga de manera automática.

INICIO

Una vez hayamos descargado con éxito el software, podremos interactuar con él. Sin salir de la terminal podemos escribir el comando “rasa init”. Esto hará que Rasa active un nuevo chatbot ya entrenado que podrás guardar en un nuevo directorio de tu ordenador. Esto es muy útil, pues sirve de primera toma de contacto. Aprovechando este nuevo bot, podemos abrir la carpeta en un programa visualizador de código (como Visual Studio Code) para comenzar a analizar los componentes de un bot en Rasa.

ESTRUCTURA GENERAL

Un chatbot de Rasa cuenta con los siguientes componentes internos:

1. Una carpeta .rasa con el código e información necesarios para que funcione el framework.
2. Una carpeta llamada actions, de la que resulta especialmente importante la carpeta action.py; es ahí donde se registran y codifican las custom actions.
3. Una carpeta caché.
4. Una carpeta llamada data. Esta alberga, a su vez, los archivos .yml de nlu, rules y stories.
5. Una carpeta models, donde se carga el nuevo modelo de rasa cada vez que lo entrenas.
6. Una carpeta tests, donde puedes configurar las test stories en el archivo .yml.
7. Un archivo yml de configuración.
8. Un archivo yml con las credenciales.
9. Un archivo yml de domain, fundamental para recoger las respuestas del bot.
10. Un archivo yml endpoint, con la dirección host en la que se cargarán las custom actions.

DATA

NLU

En este archivo yml recogemos los intents. El intent es la acción esperada del usuario del bot, una secuencia textual o numérica que active la respuesta automática del bot.

A la hora de configurar un intent, ha de seguirse el siguiente esquema:

- intent: saludo\_usuario

  examples: |

    - Hola

    - Buenos días

    - Buenas tardes

El nombre del intent es lo que se escribe a continuación de los primeros dos puntos. No admite espacio, por lo que estos deben reemplazarse por barras bajas, y es muy recomendable que refleje el propósito del intent.

La pleca representa la existencia de un conjunto de strings que pasan a definirse justo debajo. Puedes escribir tantos ejemplos veas necesarios, de tantas formas como se te ocurra para que el bot pueda identificar el intent deseado en cualquiera de los casos.

RULES

Las reglas son códigos cerrados de acción-reacción. Supone la activación automática de una determinada respuesta preconfigurada en cuanto el bot detecto un intent específico. No conviene abusar de ellas, pues es fácil que se producción confusiones y entren en conflicto dos reglas.

La estructura de una regla es la siguiente:

- rule: decir de nada

  steps:

  - intent: gracias

  - action: de\_nada

STORIES

Las stories son piezas valiosísimas a la hora de ajustar el flujo conversacional del chatbot. Sirven de entrenamiento para que el bot pueda actuar con mayor precisión en las conversaciones. Se utilizan para fijar secuencias conversacionales encadenando intents y respuestas. Una story podría ser, por ejemplo, pedirle al bot que te dé la hora.

El esquema sería el siguiente:

- story: ask for time

  steps:

    - intent: greet

    - action: utter\_greet

    - intent: give\_time

    - action: action\_show\_time

El nombre de la story es identificativo del propósito de esta. Los dos primeros pasos no guardan relación real con el objetivo de esta story, pero ayudan en la tarea de NLU por parte del bot para que comprenda mejor la estructura natural de una conversación de persona a persona.

Cuantas más stories incluyas para cada caso, mejor funcionará el bot.

TEST STORIES

Son útiles para comprobar que un mensaje ha sido clasificado correctamente como intent y para predecir las acciones. Se valen de un formato muy similar al de las stories, con la diferencia de que estas permiten además incluir al user en el código para especificar el texto del intent que el bot ha de reconocer, así como las entidades del mismo.

Un ejemplo de test story es el siguiente:

- story: bot challenge

  steps:

  - user: |

      are you a bot?

    intent: bot\_challenge

  - action: utter\_iamabot

DOMAIN

En este archivo encontramos varios elementos de interés.

Por un lado, debemos agrupar ahí todos los intents creados en el apartado de nlu.yml, a modo de lista, escribiendo únicamente el nombre elegido para el intent.

También se hace lo mismo con las custom actions, desplegando una lista de estas bajo el nombre actions.

Es en este archivo donde reflejaremos también las respuestas del bot, con su nombre y el texto que deseamos que comunique:

responses:

  utter\_greet:

  - text: "¡Hola!¿Qué tal?"

CUSTOM ACTIONS

ACTIONS

Para activar las custom actions debemos codificarlas en el archivo actions.py con Python. Estas acciones necesitan un servidor host para activarse y almacenarse ahí. Un ejemplo de custom action es el siguiente:

import datetime as dt

from typing import Any, Text, Dict, List

from rasa\_sdk import Action, Tracker

from rasa\_sdk.executor import CollectingDispatcher

class ActionShowTime(Action):

     def name(self) -> Text:

         return "action\_show\_time"

     def run(self, dispatcher: CollectingDispatcher,

             tracker: Tracker,

             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        dispatcher.utter\_message(text=f"{dt.datetime.now()}")

        return []

Las cuatro primeras líneas escritas de código sirven para activar las diferentes funciones que queremos que utilicen las custom actions. Después de class podemos definir un nombre identificativo para la acción, que ha de reflejarse después de return.

Para que funcionen necesitamos activar el siguiente código en el archivo endpoint.yml:

action\_endpoint:

  url: "http://localhost:5055/webhook"

Una vez tengamos definidas todas las custom actions en actions.py y las hayamos recopilado en el archivo domain.yml, ejecutaremos en la terminal “rasa run actions”, lo que activará las acciones personalizadas y las almacenará en la dirección http que hemos activado en el endpoint.

Una vez hecho esto solo falta entrenar de nuevo el bot con “rasa train” y probar su funcionamiento con “rasa shell”.

REGULAR EXPRESSIONS

Las regular expressions son útiles para identificar entidades. Son utilizadas para casos específicos en los que se quiere extraer entidades asociadas a fórmulas variables como correos electrónicos, direcciones o números de teléfono.

Para ello, hay que seguir los siguientes pasos:

1. Primero, crearemos un nuevo intent en el apartado nlu.yml en el que especificaremos la entidad de la que queremos extraer la expresión regular. Para ello, hay que hacer lo siguiente:
2. - intent: my\_account\_number
3. examples: |
4. - mi número de cuenta es [7987256341]{"entity": "account\_number"}
5. - [7987456321]{"entity": "account\_number"}
6. - aquí está mi número de cuenta [7123456789]{"entity": "account\_number"}
7. Una vez hayamos descrito el intent, pasamos a describir la regular expression en el mismo archivo nlu.yml.

- regex: account\_number

  examples: |

    - \d{10,12}

    - ^7\d{10,12}

Hay que tener en cuenta que el nombre de la entidad que se ha descrito en el intent ha de coincidir con el nombre de la regex. En los ejemplos de la regular expression estableceremos una secuencia de código que variará en función de lo que se desee extraer. Para más información al respecto, recomiendo la página <https://www.w3schools.com/python/python_regex.asp>.

1. Una vez hecho esto, en el archivo domain.yml debemos crear una nueva lista llamada entities (de la misma manera que creamos una al principio llamada intents) donde escribiremos el nombre de las entidades que hayamos definido.
2. Crearemos stories con el nuevo intent, incluyendo la entidad que queramos extraer. Ejemplo:

- story: cuenta

  steps:

    - intent: greet

    - action: utter\_greet

    - intent: credit\_account

      entities:

        - account: "saldo"

    - action: utter\_credit\_account

    - intent: my\_account\_number

      entities:

        - numero: 71234567891

    - action: utter\_my\_account\_number