

Intelligent Chatbot System based on Entity Extraction using RASA NLU

Raj Sinha
Software Developer
Wipro Limited, Bengaluru

Pushpendra Kumar Singh
Customer Account Manager, IT.
Tata Steel, Jamshedpur

Abstract—This paper is focusing on creating a chatbot from the RASA framework using custom components in the pipeline which is to be used by: 1) clients to get their queries responded easily from the telegram or website. The Enquiry Chatbot has the capacity to make friendly conversations; respond to the service and pricing details; give the link for the previous projects completed; answer the frequently asked questions; ask the clients for the budget and based on the client's input; and ask for the sample software which is to be made. Also a covid-19 Tracker is made and integrated in Telegram which will answer the queries about Covid-19 and send emails about the covid-19, measures, precautions in a pdf to them after taking their phone numbers and email IDs. Maximum chatbots lack empathy and fail to understand anything which is out of the script. In order to address these problems, we used the RASA framework and implemented it in the telegram. Chatbot extends the implementation of the current chatbots by adding sentiment analysis and active learning. Upto some extent sentiment analysis can recognize the user's query as positive, negative and neutral, but the system was partially successful in adding empathy to the chatbot so that it can understand unscripted queries also. It is because the system requires more rigorous training data to handle all queries which are off script. However, for such queries, active learning helps to improve the chatbot performance since it correctly understands the user's questions, asks clarifying questions, and then retrains the system to give the response what the user intends to get. In this project, we will look into how, the recently released Rasa Core, which provides machine learning based dialogue management, helps in maintaining the context of conversations using machine learning in an efficient way.

Keywords—RASA, Natural Language Processing, Natural Language Understanding, Chatbot.

I. INTRODUCTION

A chatbot is an artificial Intelligence software that simulates human conversation or "chatter" through text or voice interactions in natural language through various means such as websites, mobile apps, or the telephone. At the heart of chatbot technology lies NLP (Natural language processing), which forms the basis of the voice recognition systems used by virtual assistants such as Google Assistant, Apple's Siri, and Microsoft's Cortana. We can use both text and speech to communicate with it. Nowadays almost every open-source library is designed with the high standard of professionalism which also leads to the extension of machine learning algorithms implementations. Since RASA is also an open-source project, rasa nlu and core aim to fill the gap between research and application. Usually, the code produced by research groups falls short of expectations as to maintain a widely used project a large amount of non-

research work is involved. Thus, RASA is making technology easy by bringing recent advances in machine learning to non-experts who want to implement Conversational AI Systems. Rasa NLU and core are easy-to-use tools intended to give non-specialists a widely-used statistical dialogue system. Which resulted in RASA gaining popularity among thousands of developers worldwide. Its tools are splitted into 2 parts 1. Natural Language Understanding(NLU) and dialogue management (Rasa Core).

II. LITERATURE REVIEW AND ARCHITECTURE

RASA consists of two components,

RASA NLU: It is a natural language processing tool for classification of intent and extraction of entity from the user input and it helps the bot to understand the words of the user.
RASA Core: It's a chatbot framework with machine learning-based dialogue management which takes the structured input from the NLU and predicts the next best action using a probabilistic model like LSTM neural network.

The message so received (from user) is passed to an Interpreter and gets converted into a dictionary which includes the original text, the intent, and existing entities. This part is handled by NLU. To keep track of conversation state, an object called Tracker is implemented. This receives the information on a new message arrival. Current status of the tracker is then sent to the Policy. Further actions are decided by the Policy. The action decided by the Policy is then logged by the Tracker. User then receives the response.

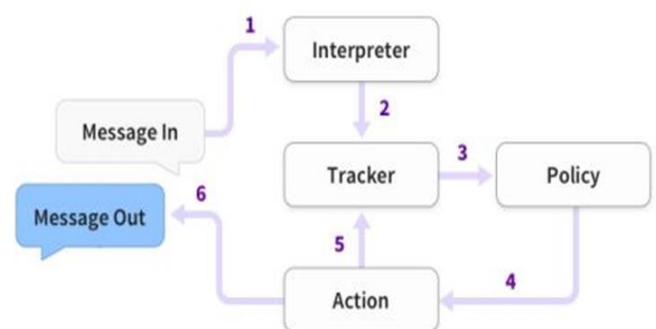


Fig.1. This diagram depicts the basic steps of how an AI assistant built with Rasa remits to a message

2.1 Dialog Flow

Dialogue elements are common conversation patterns. We have used three distinct levels of abstraction to discuss AI assistants.

- highest level: user goals
- middle level: dialogue elements
- lowest level: intents, entities, actions, slots, and templates.

The word intent is used by some chatbot tools to refer to the user's goal. This is confusing because only some messages tell us what a user's goal is. If a user says, "I want to open an account" (intent: open account), that is clearly their goal. But most user messages ("yes", "what does that mean?", "I do not know") aren't unique to one goal. In Rasa, every message has an intent, and a user goal describes what a person wants to achieve.

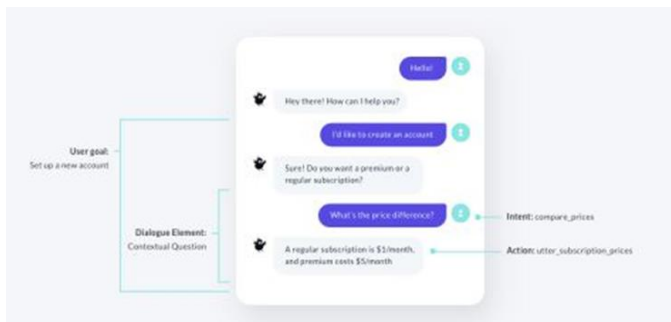


Fig.2. Understanding Dialogue Pattern

2.2 Processing RASA pipeline

In RASA NLU, incoming messages are executed one after the other through a pipeline. They are the components for intent classification and entity extraction. The processing stages which the input messages have to pass is defined by a processing pipeline. These stages are a tokenizer, featurizer, named entity recognizer, intent classifier, etc. Various pre-configured pipelines which can be used by setting the configuration values as spacy_sklearn, mitie, mitie_sklearn, keyword, tensorflow_embedding are provided by RASA.

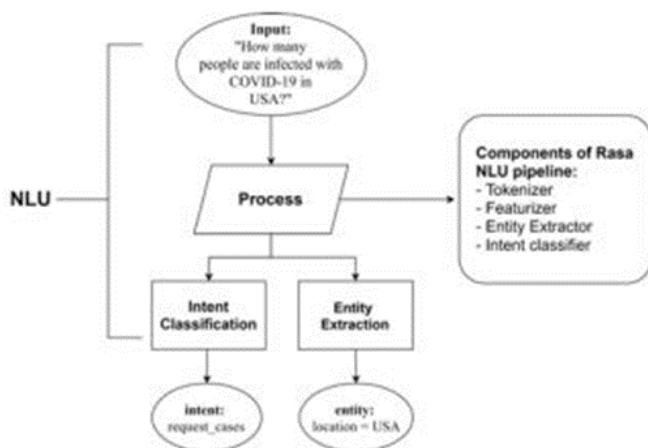


Fig.3. Schema of intent classification and entity extraction using Rasa NLU.

2.3 Policies

Rasa Core provides a class `rasa.core.policies.policy`. Current state of the tracker is sent to the Policy. Further actions are decided by the Policy. There are multiple rules based and machine learning based policies.

Type Policy	Name policy	Featurization
Machine learning policy	TED policy – the transformer embedding dialogue	The policy concatenates the user input, system actions and slots.
	Memoization Policy	The policy remembers the stories from the training data. It checks the matching story of the current conversation and predicts the next action from the matching story with the confidence of [0,1]. Number of turns of the conversation is indicated in <code>max_history</code> .
	Augmented Memoization Policy	It remembers examples from matching story for up to <code>max_history</code> turns. It is similar to the Memoization Policy. In addition, the policy has a forgetting mechanism.
Rule-based Policies	Rule policy	A policy handles conversation parts that follow a fixed behavior and makes predictions using rules that have been in the training data.
Configuring Policies	Max History	It controls the number of dialogue history that model looks at to predict the next action.
	Data Augmentation	It determines how many augmented stories are subsampled during training.
	Featurizers	It provides to apply machine learning algorithms to build up vector representations of conversational AI.

Fig.4. There are rule-based and machine-learning policies

2.4) actions.py (Custom Actions):

The utter action is defined by adding an utterance template to the domain file but if we need to run any code we want then we would need custom actions. Custom actions can do anything that you can imagine like turning on the lights, adding an event to a calendar, checking a user's bank balance, etc. Whenever, a custom action is predicted, Rasa Core calls an endpoint which has to be specified by us. This endpoint has to be a web server that reacts to this call, runs the code and optionally returns information to modify the dialogue state. To be specific, our action server we use the endpoints.yml and pass it to the scripts using --endpoints endpoints.yml

III. METHODOLOGY AND WORKING

In this project, we propose a method to improve chatbot's natural language understanding by using the custom components in pipelines. In addition, a chatbot's performance depends on how well the parameters are tuned in the policies that have been provided in Rasa Core. Rasa has pre-configured pipelines which we review for building the chatbot: TensorFlow-based pipeline, ConveRT pipeline, BERT-based pipeline. Chatbots can be built from scratch using Tensorflow pipelines. No pre-trained word vectors are used. It supports any language that can be tokenized. While using a custom tokenizer for specify-language, the "tokenizer_whitespace" can be replaced with our tokenizer with more accuracy.

ConveRT pipeline is a template pipeline which makes use of a ConveRT model for extracting pre-trained sentence embeddings. ConveRT pipeline shows the accomplishments of big training data. We also considered a pipeline which makes use of the state-of-the-art language model BERT. A pipeline with configuration for BERT using Hugging Face model is provided by Rasa. Inside the pipeline, this configuration with BERT model is possible.


```
pipeline:
- name: HFTransformersNLP
  model: "bert-base-multilingual-cased"
- name: LanguageModelTokenizer
- name: LanguageModelFeaturizer
- name: CountVectorsFeaturizer
- name: DIETClassifier
```

Fig.5. Pipeline using multilingual BERT model

We used ngrok which provided a real-time web UI where we introspected all HTTP traffic running over tunnels

While ngrok is running, it will present a series of information from the current session. We copied the url of the Forwarding field with the HTTPS protocol and pasted it in the credentials.yml under the telegram section

TELEGRAM_WEBHOOK = link_do_ngrok/webhooks / telegram / webhook



```
Command Prompt - pip install rasa
Microsoft Windows [Version 10.0.18362.959]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\RAJ>activate rasa
Could not find conda environment: 'rasa'
You can list all discoverable environments with 'conda info --envs'.

C:\Users\RAJ>activate rasa
(rasa) C:\Users\RAJ>pip install rasa
```

Fig.6. Creating RASA environment and Installing Rasa

3.1 Data Folder

This consist of two files nlu.md and stories.md.

A form of training data are RASA stories used to train Rasa's dialogue management models. A story can be understood as a depiction of a dialogue in between a user and an AI assistant, converted into a unique format where user inputs are depicted as respective intents (and entities where necessary) whereas the feedback of an assistant are depicted as respective action names. A story is a training example for the Rasa Core dialogue system. This is what is called a guide to the story data format.

(A) nlu.md : Our NLU training Data

(B) stories.md : Our stories

A story starts with a name preceded by two hashes ##story_name. The conclusion of a story is represented by a newline, and then a new story starts anew with ##. Dialogues sent by the user are shown as lines starting with * in the format intent {"entity1": "value", "entity2": "value"}. Actions carried out by the bot are shown as lines starting with - and contain the name of the action. A story starts with a name preceded by two hashes ##story_name. Events remitted by an action are on lines immediately after that action. For example, if an action returns a slotset event, this is shown as slot {"slot_name": "value"}

```
##happy path
* greet
- utter_greet
* contact_sales
- sales_form
- form{"name":"sales_form"}
- form{"name":null}
- utter_slots_values
* thankyou
- utter_noworries
```

user request
activate sales form
run form action
deactivate form

Fig.7. Sample Intents and stories (RASA_NLU & RASA_Core)

Domain.yml : The domain consists of five key parts consisting of intents, entities, slots, actions, and templates. We have already discussed the first two previously, let's understand the rest.

- slots: slots are basically bot's memory. It is a key-value storage which can be used to store data and information the user provided (e.g. their location) as well as information and data gathered about the outside world (e.g. the result of a database query).

- actions: bot's response to user input is the action. There are 3 types of actions in Rasa Core: default actions, utter actions & custom actions

- templates: templates are messages that the user gets sent back by the bot.

Following intents were used in the project

- corona_status • india • corona • karnataka • mumbai • bihar • delhi • punjab • asam • andhra pradesh • us • world • jharkhand • odisha • west bengal • tripura • kerala • meghalaya

We can edit stories.md file for more

For covid19 chatbot we are using external open source API <https://api.covid19india.org/data.json> following are the steps to build basic chatbot

Edit domain.yml and add action actions:

- action_corona_status
- action_hello_world action_search_restaurant Edit action.py class ActionCoronaTracker(Action): 31

```
def name(self) -> Text: return "action_corona_status"
```

```
def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
```

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

Commands

- rasa train (for training model)

- rasa shell (command interface provider between user and bot).

- rasa shell --debug. - rasa x (UI interface).

- rasa train nlu(training nlu.md file).

To run server - sudo rasa run actions

To run shell - sudo rasa shell

IV. RESULTS AND DISCUSSIONS

From the work it is concluded that rasa core features like slots, forms, supervised interactive learning, API integration, and database makes it a complete framework that can be used to perform highly complex tasks. The chatbot based on rasa has more capabilities than any open-source alternative. All internals and custom actions have been studied. A RASA chatbot is able to learn the intent of the user, interact intelligently, perform actions if users ask so, provide an efficient learning mechanism and most importantly doesn't use any paid services and is a completely open-source framework. Also, the true potential of the rasa NLU and core shines when we give it more data to train. The codes have been pushed to my GitHub repository (<https://github.com/RajSinha77/Covid-19-Tracker-ChatBot>) for making it an open-source project and for future additions in the working

REFERENCES

- [1] <https://kevincurran.org/dissertations/2018%20Thesis%20Dana%20Doherty%20-%20Chatbots.pdf> pg -08
- [2] "Why people use chatbots" by Petter Bae Brandtzaeg and Asbjørn Følstad, in 4th International Conference on Internet Science, 22-24 November, 2017, Thessaloniki, Greece.
- [3] "Question Answering in Restricted Domains: An Overview" by Diego Mollá and José Luis Vicede in Computational Linguistics 33(1):41-61 · March 2007.
- [4] "Advances in natural language processing" by Julia Hirschberg & Christopher D Manning in Science 349(6245):261-266 · July 2015
- [5] "A Phrasal Approach to Natural Language Interfaces over Databases" by Michael Minock in 10th International Conference on Applications of Natural Language to Information Systems, NLDB 2005, Alicante, Spain, June 15-17, 2005, Proceedings
- [6] RASA documentation Introduction to RASA